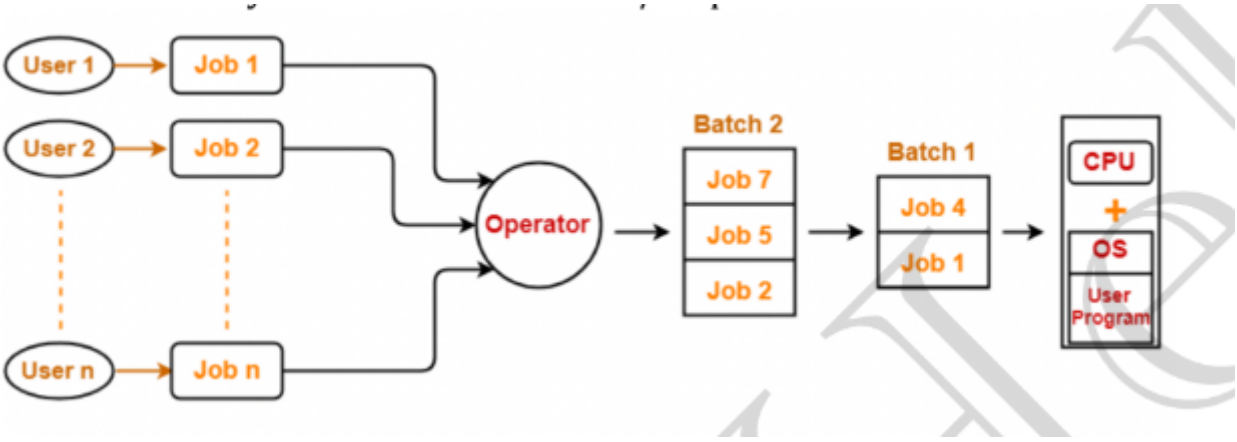
Types of operating systems:

1) batch processing operating system

ده كان زمان ايام ال mainframes اللي هو الكومبيوتر اللي قد الدولاب

1. ال user كان بيحط البرنامج بتاعه على حاجة اسمها punch cards
2. بيوديها ل شخص مسئول (انسان حقيقي) اسمه ال operator , اللي بيجمع ال programs من كذا user مختلفين بعدين يشوف ال programs اللي شبه بعض مثلا ويحطهم مع بعض ف دفعة واحدة أو batch , بعدين بيدخل الدفعة دي مرة واحدة لل mainframe اللي بعدين ب execute كل job ف الدفعة دي واحد واحد



2) multi-programming

من اسمه يعني كذا program مع بعض , ده بكل بساطة بي utilize ال processor أفضل , ازاي ؟؟؟؟
ده كان single CPU عادي , لما process تعطل على i/o مثلا بدل ال CPU م يقعد جنبها فاضي ال OS بيطلعها تستنى برة و يدخل process تانية

3) multi-tasking

ده شبه ال multi-programming شوية, وكان single CPU برضه بس مكنش بيسيب ال process براحتها على م توقف هي ,لأ كان بيحدد time-slot معينة وليكن افتراضيا 2 seconds اول م process تخلصهم يدخل التانية علطول وهكذا كل واحدة ثانيتين لحد م يخلصوا هم الاتنين

بيقولوا عليه time-interactive لان انت ك user غلبان فاكر انهم بيحصلوا مع بعض ف نفس الوقت (لكن هو processor واحد زي م قولنا) اللي بيحصل فعلا انه بيبدل بينهم بسرعة جدا فانت تحس ان الاتنين شغالين معاك ف نفس الوقت لكن هم sequential عادي ,واحدة تطلع التانية تدخل

4) multi-processing

من اسمها برضه يعني هنا عندنا اكتر من CPU ف ممكن فعلا 2 processes يشتغلوا مع بعض عادي parallel

5) real-time OS

شايف ف اللي فاتوا كلهم كان ممكن process تتأخر شوية على ما ت run لكن هنا الكلام على strictly timing constraint ,
يعني لو طلبت ال process دلوقتي لازم ال response دلوقتي والا تخيل مثلا جهاز medical زي الليزر بتدوس عليه عشان توقف راح هنج معاك , ده كلام برضو !!!
فيه hard دي اللي مفيهاش هزار , دلوقتي يعني دلوقتي زي ال medical كدة
و فيه soft يعني هو مهم بس ممكن نستحمل التأخير حبة زي ال online games مثلا ال timing مهم اه بس لو حصل delay مفيش أضرار كبيرة يعني

---

computer system startup

لما بتعمل power on للجهاز ايه اللي بيحصل بالظبط؟؟
اول حاجة ال CPU بيشتغل لأنه هو اللي بينفذ كل حاجة بعدين بيدور على برنامج اسمه bootstrap موجود على ال motherboard في ROM (read only memory) معمولة عشان اقرا منها ال بس لكن مش بنكتب فيها حاجة

ال ده بقا بي check على ال hardware ولو كله تمام بعدين يبدأ يشغله

1. initialize CPU registers , device controllers, memory

كدة ال hardware جاهز و ال ram اشتغلت , يبدأ ال bootloader بقا ي load ال kernel ف ال ram عشان نبدأ شغل
2) load kernel

ال kernel هتبدأ شغل و هتعمل load لأول process خالص اللي اسمها init اختصارا ل initialization واللي بت create بقية ال processes اللي ال system محتاجها عشان يقوم

3. kernel starts the first process and waits for an event to occur

دلوقتي ال computer بقا جاهز لل user processes.

interrupts

ال interrupt in general يعني يبقى البروسيسور ب execute some process اروح اقاطعه و اخلي يسيب اللي بيعمله ايا كان و يعمل حاجة تانية

1. hardware interrupt

   ده ال hardware بيبعتها لل CPU عشان يطلب منه حاجة معينة , مثال : hardware timer
   فاكر لما قولنا ان ال CPU بيدخل process ت execute ثانيتين مثلا بعدين يبدلها بالتانية , اللي بيحسبله الثانيتين بقا هو ال timer ده , اول ما الثانيتين يخلصوا يبعتلوا interrupt يقولوا بدل.

2. software interrupt

   دي ال software هو اللي بيبعتها لل CPU. مثلا program بي execute عادي بعدين احتاج انه ي allocate memory فبيبعت لل CPU عشان ينبهه انه محتاج ال OS يحجزله memory

ال CPU اول ما بيجيله ال interrupt سواء hw أو sw دي بيسيب ال process اللي كان بيعملها ويروح يشوف ال interrupt ده مين اللي باعته ,عرف خلاص , بيروح ل حاجة اسمها vector table

ايه ال vector table ؟؟

Interrupt Vector	Interrupt Type	Description	Handler Address
0x0E	**Floating-Point Exception**	Triggered by an error in floating-point operations, such as divide-by-zero.	0x1200
0x14	**External Interrupt**	Raised by external events like a mouse click, keyboard input, or hardware signals.	0x1400

ال CPU عرف ان اللي عمل ال interrupt هو ال mouse مثلا ف يروح لل vector table يدور على ال interrupt mouse لقاه ف ال row رقم اتنين يقوم رايح لل address 0x1400 هيلاقي هناك كود ثابت مبيتغيرش اسمه ال interrupt service routine يروح ينفذه , بعدين يرجع تاني يكمل البروسيس اللي كان بيعملها

ملحوظة أخيرة ال software interrupts نوعين بس:

1. trap

   من اسمها كدة فخ أو حفرة ● ال processor بيقع فيها غصب عنه تلاقي مثلا بي execute عادي خالص , يلاقي نفسه ب ي يقسم على 0 او بي access wrong address او اللي احنا عارفينها (segmentation fault)

2. system call هنتكلم عنها

interrupt handling

محتاجين الأول نتكلم عن حاجة اسمها **context switching**
تخيل معايا انت كنت شغال على اللاب بتاعك , وجه اخوك يقومك بسرعة وبيقولك انا محتاج اللاب ضروري حالا , قبل م تقوم وتسيبله اللاب لازم ت save اللي كنت بتعمله صح؟؟؟؟ هو ده بقا ال context switching

لما ال process تبقى شغالة و ييجي interrupt عايز ينفذ interrupt service routine تانية لازم الأول اعمل save لكل حاجة عن ال process الحالية قبل م اروح انفذ ال التانية , عشان لما ارجعلها تاني ابدأ من مكان ما كنت واقف مش من الأول خالص

---

system calls

احنا عارفين انه من أهداف ال OS انه ي manage ال resources و يحميها , تخيل مثلا لو الموضوع مفتوح عادي كدة اي process تعمل اللي هي عايزاه , كان ممكن بكل بساطة process تاخد ال CPU ليها لوحدها متسيبهوش أو مثلا process عملتها بالغلط تمسحلك ملفات من ال OS توقع الدنيا كلها مهي ليها direct access on hardware بقا

ال OS بيتحكم ف ال CPU عن طريق modes 2 لل CPU وهو اللي بيقرر انهي mode يشتغل دلوقتي

1. user mode
   هنا بيسيب لل CPU مساحة صغيرة يلعب فيها براحته , يعمل الحاجات البسيطة شوية arithmetic operations, ي push حاجة ف ال stack كدة يعني طالما مش direct access على ال hardware
2. kernel mode
   هنا بيتقال عليه ال privileged instructions أو unrestricted mode , عايز ا create process , اعمل access على hardware زي اني اخد input من ال keyboard مثلا

طب افرض ال CPU ماشي ب execute الحاجات اللي في البسيط منه اللي هو ال user mode يعني و احتاج حاجة مش ف صلاحياته و ف صلاحيات ال kernel mode ؟؟؟

محتاجين ننقل لل kernel mode صح ؟؟؟ طب نعمل ايه ؟ نستأذن ال OS عشان ينقلنا هناك , طب ازاي كنا بنطلب حاجة ؟؟؟؟ ايوة انت صح interrupt

بس ثانية ؟؟ هعمل ال interrupt ده ازاي ك developer غلبان
هنا بقا ييجي دور ال system call

ال system call عبارة عن function او API بتعملها call ف ال program بتاعك
ال function دي ال implementation بتاعها فيه شوية assembly كدة اول ما ال CPU بيشوفه بيتعمله software interrupt علطول
ف يقوم ال OS ينقله لل kernel mode و هنا بقا كل حاجة مسموحة طبعا ف يروح لل vector table ويهندل اللي ال sys call كانت محتاجاه زي مثلا انه ي allocate memory
زي scanf مثلا

CPU scheduling

قبل ما نشرح ال CPU scheduling لازم نتعلم الأول ازاي نشتغل ف سايبر

نرجع بالزمن شوية تحديدا 2005 مكنش فيه كومبيوترات ف كل بيت زي دلوقتي و كان فيه سايبر فيه

1. كومبيوتر واحد
2. واحد ماسك الشيفت

عندنا طابور واقف على باب السيبر ده مكون من

1.) محاسب عايز يشتغل على شيت اكسيل مهم جدا (ساعتين)
2. ولد صغير (شقي) جي يلعب spiderman ----------> (ساعة)
3. رجل اعمال (ف بداية حياته) محتاج يشوف ايميل ضروري ----> (5 دقايق)
4. طالب جاي يحمل بحث 10 ورقات (3 دقايق)

واقفين بالترتيب ده

عندنا 4 بيوقفوا ف السايبر ده

👴 اللي ماسك الشيفت أي حد بييجي يقف ف الطابور بيطلب منه

- اسمه أو ال ID
- محتاج يعمل ايه
- هيقعد اد ايه estimated يعني

وبيسجلهم عشان يعرف يحاسبه (وينظم الدنيا برضو) , الورقة اللي بيسجل فيها, اسمها process control block

عندنا 4 بيبدلوا الشيفتات تعالى نتعرف عليهم

1) first come first serve

rule

الراجل ده مش بيقول غير جملة واحدة "بالدور يجماعة"

يعني اللي جه الأول هو اللي هيقعد على الكمبيوتر الأول
ف هيدخل الطابور بترتيبه عادي
المحاسب ثم الولد ثم المدرس ثم رجل الأعمال ثم الطالب

disadvantages

ده جدول اليوم
السيبر فتح الساعة 00:00 , كانوا كلهم واقفين

waiting time	end	start	
0	02:00	0:00	المحاسب (2h)
2 hour	03:00	02:00	الولد الصغير (1h)
3 hours 5 min	03:05	03:00	رجل أعمال (5m)
3 hours 8 min	03:38	03:35	الطالب (3m)

total waiting time = (0 + 2 + 3.xx + 3.xx) / 5 = 2. hour

من غير كلام معادلات

1. الناس مش بتحب الراجل اللي ماسك الشيفت ده عشان انت لو عايز مصلحة , قدامك حولي ساعتين هتقف تستنى

الناس بيحبوا اللي بيمسك الشيفت بعده أكتر , تعالى اعرفك عليه

shortest job first

rule

الراجل ده بيبص عنده يشوف مين اللي هيستخدم الجهاز أقل وقت و يدخله

waiting time	end	start	
0	00:03	0:00	الطالب (3m)
3 min	00:08	00:03	رجل أعمال (5m)
8 min	01:08	00:08	ولد صغير (1h)
1 hour 8 min	03:08	01:08	المحاسب (2h)

average waiting time = 0 + 3 min + 8 min + 68 min / 4 = 20 min

adv

احسن ال algorithm تاخد منه أقل average waiting time

priority scheduling

rule

فاكر لما قولتلك ان أي حد بيقف ف الطابور بيطلب منه شوية بيانات (PCB)
الراجل ده بيقيم الطلب بتاع كل واحد , و يدي كل واحد (ترتيب من عنده حسب تقييمه لطلباتهم)

وهيشوف ان رجل الأعمال ده حاجته urgent و مفروض تتعمل قبل الطالب اللي جي يعمل البحث , مع ان الطالب هيخلص ف وقت أقل

shortest job ❌

ف هيشوف مثلا ان المحاسب ده راجل محترم بيعمل شغل مهم 😨 فهيدخله قبل الولد الصغير اللي داخل يلعب 🤺 مع ان ممكن يكون الولد الصغير جه الأول

first come ❌

priority	
2	الطالب
1	رجل الأعمال
3	المحاسب
4	الولد الصغير

waiting time	end	start	
0	00:05	0:00	رجل الأعمال (5m)
5 min	00:08	00:05	الطالب 3m
8 min	02:08	00:08	المحاسب 2h
2 hour 8 min	03:08	02:08	الولد الصغير 1h

لحد هنا الدنيا لذيذة , فيه ملحوظة صغيرة

1. أول واحد اللي هو (first come) الناس مش بتحبه اه عشان بيخليهم يستنوا كتير لكن فيه ميزة ان علطول رايق ولو قعدت على الجهاز مش بيقومك لحد م تخلص

مود الروقان ده اسمه non preemption يعني (مش متضايق بالانجلش) (ف المعجم الايرلندي كدة وكدة)

2. الاتنين اللي بعد كدة اللي هم (shortest و priority)
ممكن يبقوا

- preemptive (متضايقين)

هنا للأسف الراجل ده بيتلكك عشان يقومك
عينه علطول عالطابور , لو لمح حد جديد (أحسن أو أقصر) منك وصل الطابور هيقومك علطول

- non preemptive (مش متضايقين)

هنا بيسيبوك على راحتك لحد م تخلص حتى لو مين جه ف الطابور

تعالى نشوف واحد فيهم اللي هو shortest لما بيبقى متضايق بيعمل ايه (preemptive)

arrival time (وصل امتى)	cpu time	
0	7	p1
2	4	p2
4	1	p3
5	4	p4

لو فاكر قولنا ان عينه على الطابور علطول , ف لازم نركز ف ال arrival time

solution:

time	ready queue	cpu	shortest in (cpu, queue)	updated ready queue	next arrival time	finish time
0	p1 (7)		p1 (7)	empty	2	7
2	p2 (4)	p1 (5)	p2 (4)	p1 (5)	4	6
4	p1 (5), p3(1)	p2 (2)	p3 (1)	p1(5), p2 (2)	5	5
5	p1(5), p2 (2), p4 (4)		p2 (2)	p1(5) , p4(4)	X	7
7	p1(5), p4(4)		p4(4)	p1(5)	X	11
11	p1(5)	idle	p1(5)	empty	X	16

تعالى بقى نشرح ازاي حلينا مسئلة الpreemption

time	ready queue	cpu	shortest in (cpu , queue)	updated ready queue	next arrival time	assumed finish time
0	p1 (7)		p1 (7)	empty	2	7

1. ال time هنبدأه ب 0 اللي هي أول arrival time
2. ال ready queue عبارة عن (ال updated ready queue ف الصف اللي فات) + (لو فيه process arrived) اللي هم (empty) + (p1)
3. ال cpu (ال shortest من الصف اللي فات) يبقى empty
4. أقصر process عندي ف السيستم كله (بقارن بين ال cpu cell و ال ready queue cell)

- ال cpu ---> فاضي
- ال ready queue فيه p1(7)

5. ال updated ready queue فيه اللي خسروا المقارنة (محدش خسر عشان كان عندي p1 بس)
6. ال next arrival time بجيبها من الجدول اللي فوق ----> 2
7. ال process اللي عندي ف الcpu مفروض تخلص امتى لو محدش جه ------> 0 + 7

> شايف رقم 6 ورقم 7 شوف أصغر time فيهم و ابدأ بيه ال row الجديد

time	ready queue	cpu	shortest	updated ready queue	next arrival time	assumed finish time
2	p2 (4)	p1 (5)	p2 (4)	p1 (5)	4	6

1. من ال row اللي قبله ال arrival أصغر من ال finish ف هناخد ال 2
2. ال ready queue عبارة عن (ال updated ready queue ف الصف اللي فات) + (لو فيه process arrived) اللي هم (empty) + (p2)
3. ال cpu (ال shortest ف الصف اللي فات) بس هي بقالها ثانيتين شغالة فهتبقى p1 (7-2) اللي هي p1(5)
4. أقصر process عندي ف السيستم كله (بقارن بين ال ال cpu cell و ال ready queue cell)

- ال cpu ---> فيه p1 (5)
- ال ready queue فيه p2(4)

5. ال updated ready queue فيه اللي خسروا المقارنة اللي هي p1
6. ال next arrival time من الجدول تبقى 4
7. ال finish time لو محدش جه 2 + 4 يبقى 6

> لما ال arrival time يبقى أصغر ده معناه ان فيه حد هييجي قبل م اخلص , ف تروح تجيبه من الجدول
>
> ,
>
> ف أي مسألة الخطوة 3 هي الخطوة الوحيدة اللي بنطرح فيها

هنكمل بنفس التسلسل مفيش جديد , بعد م تخلص الجدول يهمك منه العمودين بتوع ال shortest time ,
ده اللي بتلاقيه مرسوم ف ال slides

واحدة كمان على ال priority

process	arrival time	burst time	priority
p1	0	7	3
p2	2	4	2
p3	4	1	4
p4	5	4	1

time	ready queue	cpu	highest priority (cpu, queue)	updated ready queue	next arrival time	assumed finish time
0	p1		p1(7)	empty	2	7
2	p2	p1(5)	p2(4)	p1(5)	4	6
4	p1, p3	p2(2)	p2(2)	p1(5), p3	5	6
5	p4, p1(5),p3	p2(1)	p4(4)	p1(5), p3, p2(1)	X	9
9	p1(5), p3, p2(1)	empty	p2(1)	p3, p1(5)	X	10
10	p3, p1(5)	empty	p1(5)	p3	X	15
15	p3	empty	p3(1)	empty	X	16
16	empty	empty	empty	empty	X	X

تعالى نشوف حليناها ازاي

بعد ما ملينا أول row تعالى نشوف الاتنين اللي تحته:

time	ready queue	cpu	highest priority (cpu, queue)	updated ready queue	next arrival time	assumed finish time
2	p2	p1(5)	p2(4)	p1(5)	4	8

1. ال time ب 2 , عشان دي أصغر قيمة بين (ال arrival و ال finish) ف الصف اللي قبلي
2. ال ready queue قولنا ال (updated من الصف اللي فات) + (arrived process) يبقى p2
3. ال cpu قولنا ال highest من الصف اللي فات و متنساش تطرح اشتغل اد ايه
4. ال highest priority هنقارن بين الاتنين اللي قبليا اللي لسا كاتبهم يبقى p2 تكسب (الرقم الأقل هو اللي بيكسب)

- cpu ----> p1 (3)
- ready queue -----> p2 (2)

5. ال updated ready queue (هيبقى فيه اللي خسروا المقارنة) اللي هم p1
6. ال next arrival من الجدول 4
7. ال assumed finish اللي ماسك ال cpu هيخلص امتى لو محدش قطعه 4+2 = 6

- و الأصغر ما بينهم اللي هي ال 4 هنبدأ بيها الجدول الجديد (لو هي قيمة ال arrival يبقى ابص ف الجدول واجيبها)

---

round robin

ده بيخلي كل واحد يقعد عالجهاز ربع ساعة بعدين يقومه والتاني يقعد وهكذا

ف المثال اللي فات
الكومبيوتر بيمثل طبعا ال cpu
ال محاسب و الولد الصغير والباقي بيمثلوا ال ready processes
اللي ماسكين الشيفت بيمثلوا ال dispatcher
شخصياتهم المختلفة بيمثلوا ال scheduling algorithms

ال short time scheduler ده بيختار ناس من ال ready queue يديهم ال control ع الكمبيوتر
الlong time scheduling ده بيختار ناس من بقية الqueues يحطهم ف ال ready queue

what is a thread?

single threaded vs multithreaded program

تخيل معايا مطعم برجر 🍔 فيه:

1. موظف واحد شغال , بياخد ال orders وبيعمل البرجر

لو عندنا طابور فيه عشرين واحد عايزين برجر:

single thread process(الكود الطبيعي اللي احنا بنكتبه):

ال process بتاعتنا كتلة واحدة

> خد الاوردر , اعمل البرجر بعدين سلمه للعميل

الكاشير بياخد الاوردر , بيعمل البرجر يستنى 10 دقايق لما يخلص , يسلم البرجر للعميل , بعدين يبدأ ياخد اوردر من العميل التاني وهكذا , بالوضع ده فيها 20 * 10 دقايق يعني 200 دقيقة (low performance) عشان يخلص كل اللي ف الطابور.

الكود استرشادي متركزش اوي يعني

```
public static void main(String[] args) {
SingleThreadRestaurant restaurant = new SingleThreadRestaurant();
while(true)
{
        restaurant.takeOrder();
        restaurant.cook();
        // 10 minutes
}
```

الوضع صعب صح ؟؟؟
جبنا موظف ف المطبخ يشتغل معاه 😃

فبقى الكاشير ياخد الاوردر يعمل البرجر يستنى 10 دقايق جنبه لحد م يخلص , بعدين يسلم البرجر للعميل ويبدأ ياخد اوردر من عميل تاني , و موظف المطبخ واقف بيشرب شاي

ايه العبط ده 😠 ؟ طب واحنا استفادنا ايه كدة فين الموظف الجديد ؟؟؟ بالظبط مستفدناش حاجة عشان حضرتك كاتب الكود ككتلة واحدة من غير threads

> single-thread process waste resources

multi thread process

ال process بتاعتنا هنكسرها ل جزئين منفصلين ممكن يشتغلوا سوا

> 1. جزء هيبقى مسئول انه ياخد الاوردر

> 2)جزء هيبقى مسئول انه يعمل البرجر

الكاشير بياخد الاوردر , بيكتبه ف ورقة و يسيبه لموظف المطبخ , بعدين يرجع ياخد اوردر من العميل التاني فهنا الموظفين شغالين كلهم (full use of resource) وبالتالي الطابور هيخلص ف وقت أقل (higher performance)

```
Thread orderThread = new Thread(() -> { restaurant.takeOrders();});
Thread cookThread =  new Thread(() -> { restaurant.cook();});
```

ايه اللي فرق يعني 😆
ف ال single thread كان عندك عيب ف ال design انك بتستنى cook function تخلص ال 10 دقايق بتوعها عشان تقدر ترجع ف اللوب و تشغل take order function من تاني (عشان الكود بيمشي من فوق لتحت sequentially)

طب هو انا لازم اعمل البرجر واسلمه عشان اخد اوردر جديد ؟؟؟؟؟ طيب take order تستنى ال 10 دقايق دول ليه وهي مش بتعتمد على ال cook أصلا

و هنا اكتشفنا ان ال process ممكن تتقسم ل اجزاء أصغر يشتغلوا مع بعض ف نفس الوقت , مش محتاج اخلص عمل البرجر عشان اخد الاوردر , ممكن اعمل الاتنين سوا

لما ال process بتاعتك اتقسمت لجزئين أصغر (two threads)
ال OS شايف حاجتين منفصلتين محتاجين يتعملوا ف هيعملهم scheduling كأي اتنين process منفصلين ي execute دي شوية ودي شوية ,و يا سلام بقا لو عندك أكتر من processor core كل واحدة هتشتغل على core ف نفس الوقت

🚧 شوفت ازاي لما بتقسم ال process بتاعتك ل أجزاء أصغر منفصلة الموضوع بيفرق؟؟
اهو كل جزء من الأجزاء الصغيرة دي اسمه thread

Concurrency

ال concurrency ان يبقى عندي أكتر من process بيحصلهم execution مع بعض ف نفس الوقت
وده بيحصل بطريقتين

- interleaved
- overlapping

multitasking with single CPU (interleaved)

متلخبط صح ؟ ازاي ف نفس الوقت , واحنا اصلا معندناش غير هو CPU واحد

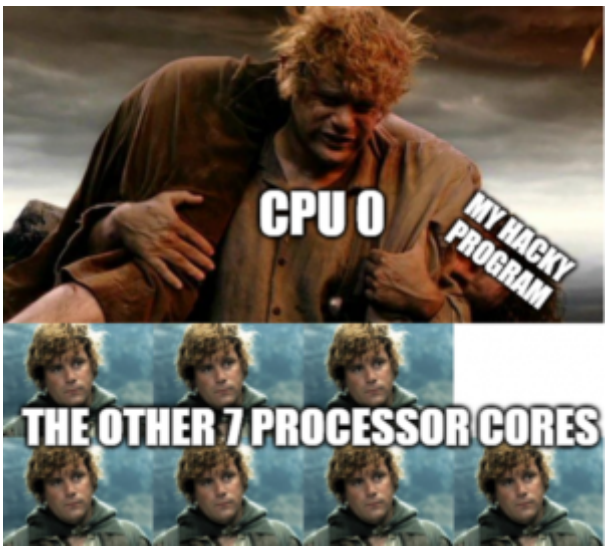
انت صح على فكرة طب ليه بيقولوا عليه multitasking ؟؟؟؟
لان ال CPU بيبدل ما بينهم بسرعة جدا ف انت مش هتلاحظ طبعا السرعة دي وهتفتكرهم بيحصلوا ف نفس الوقت

multitasking with multiple CPUs (overlapping)

ده بقا ال multitasking ال 💯 لان احنا هنا نقدر نعمل كذا حاجة ف نفس الوقت عشان عندنا كذا CPU

عايز اقولك انك لما بتكتب الكود ك كتلة واحدة من غير ال threads وانت مثلا عندك processor فيه 2 cores , ف انت معندكش غير task او process واحدة بس تتعمل ف نظر ال CPU ف هيقعد مستني ال cook تخلص عشان يقدر ي execute ال order زي م قولنا ف السيناريو الأول

> without threading , waste resources, no multitasking------>low performance



لكن لو بتكتب الكود بتاعك multithreaded ف ال OS هيعملهم scheduling بشكل منفصل ف انت لو عندك

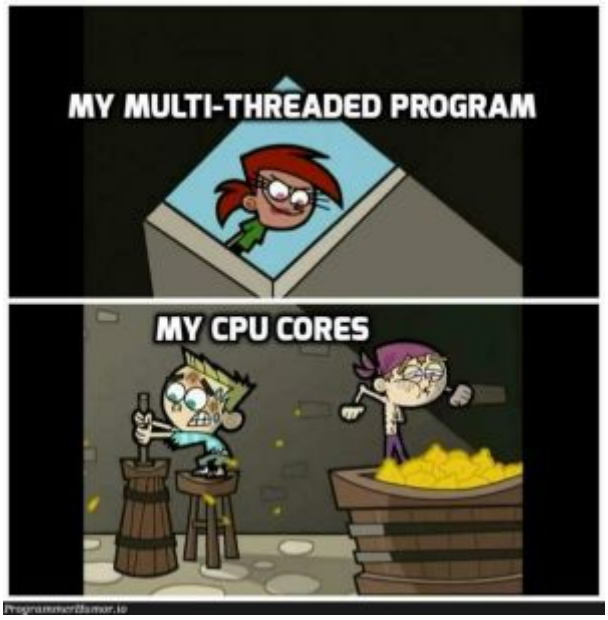
1. ال CPU (one core)
   ف انت هنا هيحصل timeshare و الوضع هيبقى أفضل على الأقل من اللي فوق, لانه شوية هيشغل take order وشوية cook وهيبدل بينهم بسرعة

كأن الكاشير ياخد اوردر ويروح يقلب البرجر شوية عالنار بعدين يرجع ياخد اوردر بعدين يرجع يحط صوص

2. ال CPU (multiple cores)
   هنا بقى عم ال multitasking بجد لأن فيه core هت execute ال take order و ف نفس الوقت ال core التانية هت execute ال cook

الكاشير واقف ياخد اوردرز بس و موظف المطبخ واقف يعمل برجر بس

> with threading , full use of resources, concurrency(multitasking) ------>high performance



اظن كدة فهمنا ليه بيتقال عليها light weight process (LWT) ------> عشان هي جزء صغير من ال process ويكأنها process مصغرة

هو فيه process من غير threads ؟
لأ ي سيدي مفيش , ال process العادية بتاعتنا دي اسمها single thread.

بيقولك بقا ان ال threads بتاعت ال process الواحدة بيتشاركوا ف حاجات كدة :

1. code segment :

يعني thread 1 بتاعت take order شايفة الكود بتاع thread 2 اللي هي cook والعكس
ايوة متفق معاك مش ميزة جامدة اوي يعني

2. data segment:

يعني ال global variables بينهم اقدر اعمل عليها access من أي واحد فيهم
عشان كدة بيقولك انهم أجمد من ال processes ف ال communication

3. files and resources

طالما ال resource بقى بتاع ال process يبقى بتاعنا كلنا 🤷‍♂️🤷‍♂️

يقولك بقا ان ال threads بتاعت ال process الواحدة بيختلفوا ف حاجات كدة أصل فين الخصوصية برضو :

1. stack

مش هي ال thread عبارة عن functions زي take order كدة يبقى لازملنا stack لكل واحد

2. CPU registers
3. TCB (thread control block)

شبه بتاع ال process كدة

---

dispatching multi-threaded processes

يا تري ال process لما تبقى running انهي thread فيها اللي هيبقى شغال ؟
منعرفش للأسف 😳 الموضوع خارج تحكمنا

ف احنا كدة ال design بتاعنا غلط 😱
لاننا اعتمدنا ان ال take order مش بتعتمد على ال cook , لكن نسينا ان ال cook بتعتمد على اني لازم اخد اوردر الأول
متخيل ي مان ان ممكن ال thread بتاعت ال cook تحصل قبل م اخد الاوردر أصلا , طب ده كلام برضو

```
Thread orderThread = new Thread(() -> { restaurant.takeOrders();});
Thread cookThread =  new Thread(() -> { restaurant.cook();});
```

solution

```
private static boolean orderTaken = false;

public static synchronized void takeOrder() {
        //take order processing..........
        orderTaken = true;
}

public static synchronized void cook() {
        while (!orderTaken) { wait(); }
        // cook processing.................
}
```

خدت بالك من ال global variables و ال thread communication والحاجات الحلوة دي
مخدتش بالك 😱 طب بص

لو ال cook جت ت execute قبل ما ناخد order هيلاقي ان ال global variable ب false ف هيستنى لحد م الاوردر يتاخد

و بكدة اتأكدنا ان الدنيا ف السليم منه 💯 بفضل ال shared memory اللي سهلتلنا ال thread communication

---

Benefits of threads

1. responsiveness

سرعة الاستجابة , زي م شرحنا فوق كدة

2. resource sharing
3. economy(full use of resources)
4. utilization of multiprocessor architecture

user level threads vs kernel level threads

user level threads

دي ال threads اللي احنا بنكتبها ف الكود بتاعنا , فعشان كدة بنقول created by user apps

ال OS ميعرفش عنها حاجة حرفيا بياخدك على قد عقلك , هو شايف بس ال process من برة واحدة بس

طب بالعقل كدة طالما مش شايفهم اومال مين اللي بيختار ال thread اللي عليها الدور عشان ت execute ؟؟؟
لما ال kernel بيختار ال process ,ال library اللي انت عاملها include ف الكود بتاعك فيها كود ال management لل threads دي كلها

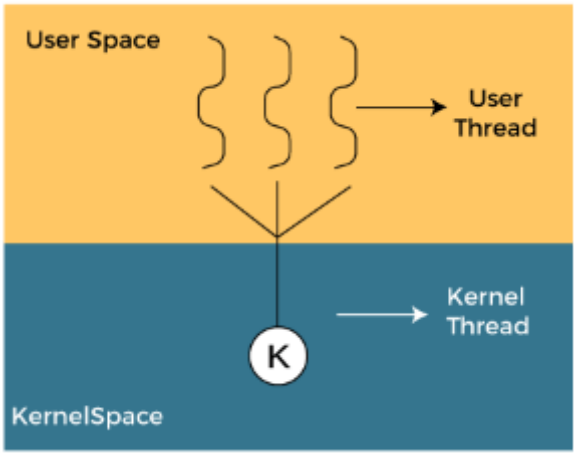
disadvantages

لو انا ك kernel شايفك process واحدة يبقى

1. مهما كان جواك threads مش هتاخد مني غير CPU واحد لاني شايفك process واحدة و ال kernel مش عبيط عشان ي allocate اكتر من core لنفس ال process , ف مفيش multiprocessing
2. لو thread عازت i/o اللي هيحصل هتقول لل kernel صح ؟ ال kernel مش شايفاها هي شايفة ال process هي اللي طلبت ال i/o ف هترمي ال process كلها على ال blocking مع ان كان فيه threads تانية كان ممكن تشتغل

kernel level threads

دي بقا ال kernel هي اللي بت create ها
طبعا ال kernel شايفها ثرداية ثرداية ف ال scheduling هنا على المكشوف عادي , ف فيه multiprocessing عادي و لو thread احتاجت i/o هعملها هي بس blocking مش ال process كلها



متخيل ي مان ال user level ضعيفة ازاي عشان كدة محتاجين نعمل حاجة اسمها mapping

mapping

M : 1

زي ما شرحنا فوق كدة ال many user threads بت map ل kernel thread واحدة ف

1. ال kernel شايف kernel thread واحدة بس ف مفيش multi processing و لو thread حصلها blocking بقية ال threads هتحصلها
2. ال user threads مش متشافة من ال kernel ف هتتهندل ب ال library

one : one

هنا كل user thread بت map ل kernel thread ف

1. ال kernel شايفهم كلهم عادي وهو اللي هيهندل ال scheduling
2. فيه multiprocessing و مفيش مشكلة لو حصل blocking ل thread
3. متخيل ان لو عندنا 4 process كل واحدة فيها 3 threads هيبقى توتال 12 user threads يعني 12 kernel threads , بيحصلهم scheduling بشكل منفصل فالموضوع مكلف

M : M (smaller)

هنا كل مثلا لو عندنا five user threads بت map لtwo kernel thread ف

1. ال kernel شايف ال 5 عبارة عن 2 بس , فهيحصل multiprocessing عادي لأني شايف 2 processes الاولى هتاخد processor والتانية هتاخد processor , ولو حصل blocking ل 1 thread يقدر ينقل على التانية مش هيعمل blocking لل user process كلها
2. ال kernel و ال library بيهندلوا ال scheduling سوا

---

mutual exclusion

عشان نشرح ال mutual exclusion محتاجين ننزل الجيم
نزلنا الجيم خلاص , حطينا الشنطة ف locker , يلا نروح نتمرن بقى ,استنى مكانك 😟 تعالى نتكلم على ال locker شوية بعدين نروح نتمرن

ال locker ده اسمه shared resource لانه مش معمول لحضرتك مخصوص يعني , ممكن انت تستخدمه , ممكن حد تاني, ممكن حد تالت لأنه public صح؟؟

مشكلة ال shared resource زي ال locker كدة , ممكن حد معندوش ضمير 👺 يجي يرمي شنطتك ف الأرض ويحط شنطته أو يحط حاجته جنبك (لو طيب يعني)

طب نعمل ايه ؟؟؟ محتاجين نتصرف ؟؟ هنا بقا ييجي ال mutual exclusion

ال mutual exclusion معناه ان محدش يستخدم ال resource قبل م التاني يخلص استخدامه

طب وهنعمل كدة ازاي ؟
💡 ادارة الجيم هتدي حضرتك مفتاح 🔑 وانت داخل , هتقفل ال locker , تخلص تمرينك , تاخد حاجتك وتسلمهم المفتاح

ال lock ده طريقة من الطرق اللي حققتلي ال mutual exclusion

بنحتاج نطبق ال mutual exclusion لما يبقى عندنا

1. shared resource
   مش ليك مخصوص , لكنه معمول للاستخدام العام ال locker
2. concurrency
   ده ممكن حد يحاول يستخدمه معاك ال locker

```c
int x = 20;
int main()
{
        func1();
        func2();
}
```

الكود ده sequentially من تحت لفوق كأننا بنقول func1 هتحط شنطتها ف الlocker وتخرج , بعدين func2 هتيجي بعدها تستعمل ال locker
ف عندنا process واحدة بس مفيش غيرها بيحصل فيها كل حاجة

shared resource ✅ (global variable x)
concurrency ❌

بالتالي مش محتاجين نحقق ال mutual exclusion

```
int x = 20;
Thread f1 = new Thread(() -> { func1();});
Thread f2 = new Thread(() -> { func2();});
```

لكن لو عندنا threads او processes 2 كوبي بيست من بعض

1. ال processor شايفهم اتنين منفصلين و هيعملهم scheduling ك أي اتنين منفصلين ف ممكن يبدل بينهم بسرعة او ينفذهم in parallel كل واحدة على core منفصل

shared resource (global variable) ✅
concurrency ✅

> بالتالي محتاجين نحقق ال mutual exclusion

what can happen if didn't achieve mutual exclusion?

عندنا اتنين أخلاقهم مش أحسن حاجة عصام و اياد , عصام قال لاياد انا معايا 100 جنيه ف البنك , هعمل log in على موقع البنك من جهازين مختلفين و ادوس transfer ف نفس الوقت وبكدة يبقى وصلك 200 جنيه يمعلم خد 100 و هات ال100 بتاعتي

process 1

```
int essam_balance = 100;
int eyad_balance = 0;

void transfer(int amount,// from essam , to eyad)
{
        if(amount >= balance)
        {
                eyad_balnce   += amount;
                essam_balance -= amount;
        }
}
```

process 2

```
int essam_balance = 100;
int eyad_balance = 0;

void transfer(int amount,// from essam , to eyad)
{
        if(amount >= balance)
        {
                eyad_balnce   += amount;
                essam_balance -= amount;
```

```
        }
}
```

انت ك server جالك ال 2 processes دول ف نفس الوقت (عشان عصام ضغط على transfer ف نفس الوقت)
جه ال scheduler بدأ ب الprocess الأولى وشال منها ال processor بعد ال timeslot
زودت اياد 100 و لسا مخصمتش المبلغ من حساب عصام

```
eyad_balance += amount; (done)
// ..to be continue in next timeslot
//essam_balance -= amount;
```

ال process 2 بدأت شغل لقت ال condition تمام راحت دخلت هي التانية و زودت اياد 100 كمان و timeslot بتاعتها خلصت

```
eyad_balance += amount;(done)
// .. to be continue in next timeslot
//essam_balance -= amount;
```

رجعت ال process 1 تكمل execution خصمت المبلغ بقى ال essam_balance = 0
رجعت process 2 تكمل execution خصمت المبلغ بقى ال essam_balance = -100

وبكدة عصام واياد ضحكوا عليك عشان حضرتك عارف ان عندك shared resource و ممكن يحصل عليه concurrency و محققتش ال mutual exclusion

how can we achieve mutual exclusion

بص على الكود كدة و قولي انهي code block اللي مينفعش حد يدخله قبل م التاني يخلصه كله ؟؟؟
دي صح

```
if(amount >= balance)
{
                eyad_balnce   += amount;
                essam_balance -= amount;
}
```

حتة الكود دي بقا ي سيدي اسمها ال critical section دي شبة ال locker كدة

هنعمل ايه بقا ؟؟؟

بسيطة هنقفلها بالمفتاح وبعد م نخلص هنفتحها تاني

```
void transfer(int amount,// from essam , to eyad)
{
        take_key_and_lock();
        if(amount >= balance)
        {
                eyad_balnce   += amount;
                essam_balance -= amount;
        }
        unlock_and_release_key();
}
```

طبعا ال functions دي مش حقيقية ولكن ال concept واحد وده بيحصل فعلا مش بضحك عليك ولكن ال functions دي بيحصلها implementations بطرق مختلفة

ايه الفرق بقا ؟؟؟

عصام ضغط transfer مرتين ف نفس الوقت زي ما هو وعمل creation ل اتنين processes

processes 1

```
void transfer(int amount,// from essam , to eyad)
{
        take_key_and_lock();
        if(amount >= balance)
        {
                eyad_balnce   += amount;
                essam_balance -= amount;
        }
        unlock_and_release_key();
}
```

بدأت process 1 زودت اياد 100 و لسا مخصمتش المبلغ من حساب عصام كالعادة كانت ال timeslot خلصت

```
eyad_balance += amount;(done)
```

بعدين ال CPU اتاخد منها وراح ينفذ process 2

```
void transfer(int amount,// from essam , to eyad)
{
        take_key_and_lock();
        if(amount >= balance)
        {
                eyad_balnce   += amount;
                essam_balance -= amount;
        }
        unlock_and_release_key();
}
```

هنا الصدمة

`take_key_and_lock()

ال key مع process 1 ي معلم , قصدي ال locker مقفول و المفتاح مع حد غيرك , استنى بقى ف طابور ال waiting على ما process 1 تخلص ,وتسيب ال key

شوفت ال mutual exclusion حلو ازاي

---

2. printer

ناخد مثال تاني عشان نثبت الفكرة

من الحاجات اللي محتاجين نعملها mutual exclusion (يعني محدش يستخدم ال resource قبل م التاني يخلص استخدامه) هي ال printer
يا ترى ليه؟؟؟؟ عشان shared resource و concurrency ممكن تحصل

تخيل مثلا process عايزة تطبع file Y و process تانية عايزة تطبع file X , الاتنين محتاجين ال printer حالا بالا فالا ينفع process تطبع سطرين من file X بعدين ال CPU يطلعها تستنى و ينزل التانية تطبع سطرين من file Y ؟؟
أكيد لأ

ال OS هنا بيقولك مش بتاعتك انت دي ده hardware من اختصاصي انا ههندل أنا ال mutual exclusion

وبيعمل نفس اللي كنا بنعمله فوق اللي هتاخد ال printer الأول هتقفل عليها لحد ما تخلص شغلها.
طب لو ال timeslot بتاعتها خلصت وهي لسا مخلصتش؟؟؟
ولا أي مشكلة هتاخد المفتاح معاها وهي طالعة, مش هتسيبه لحد ما ترجع تاني وتخلص استخدامها لل printer

كأن ال الكود كده

```
acquire_printer();
printing();
releasing_printer();
```

تيجي أي process تانية تلاقي ال printer مع حد تاني ولازم تستنى

عرفت ليه بقى ي سيدي ال mutual exclusion أحد أسباب ال deadlock ؟؟
عشان ال process حتى لو مش running ممكن يبقى معاها resource مش هتسيبه لحد م تخلص هتعمل lock عليه يعني

> if there is no lock, there is no deadlock

مش عارف يعني ايه deadlock أصلا طب تعالى

DEADLOCK

ف أحد الامتحانات , عصام و اياد قاعدين جنب بعض , عصام نسى القلم بتاعه , طلب من اياد قلم لكن اياد مش معاه
وهم بيوزعوا الامتحان قالوا كل اتنين جنب بعض ياخدوا ورقة اسئلة واحدة, وكل واحد يكتب الاجابات بتاعته ف ورقة خارجية

عصام قام نتش ورقة الأسئلة بسرعة وقال ل اياد هات القلم و الا مفيش ورق
اياد قاله هات انت الورقة والا مفيش قلم

عصام
معاه ورقة الأسئلة ✅
معهوش قلم ❌
ف مش هيعرف يعمل حاجة

اياد
معاه قلم ✅
معهوش ورقة اسئلة ❌
ف مش هيعرف يعمل حاجة برضو

💢 للأسف الاتنين مستحيل حد فيهم يتنازل عن اللي معاه لأن أخلاقهم مش أحسن حاجة زي ما شوفنا قبل كدة.

الوقت خلص و محدش حل حاجة ف الاخر

deadlock conditions

1) mutual exclusion

💢 problem discerption:

: فاكر لما قولتلك ان ال mutual exclusion يعني)مينفعش حد يستخدم resource قبل م التاني يخلص(و ال process لو عملت acquire ل resource مش هتسيبه الا لما تخلص

الحتة دي شبه أخلاق عصام واياد كدة اللي مش أحسن حاجة , يعني ممكن اي واحدة فيهم تسيب ال resource للتانية لكن ال mutual exclusion (الأخلاق اللي مش أحسن حاجة) بتحتم عليهم ميسيبوش ال resource غير لما كل واحد يخلص مصلحته

🍎 solution:

> مفيش حل لل mutual exclusion , أصل هتعمل ايه عندك printer واحدة مفيش غيرها ومش هينفع حد يستخدمها قبل م التاني يخلص.

2) hold and wait

💢 problem discerption:

من اسمها يعني كل process بيبقى معاها resource و مستنية resource
زي عصام كدة معاه القلم و مستنى الورقة و اياد مستني القلم ومعاه الورقة

🍏 solution:

عارف المراقب لو كان قاله طالما مش معاك قلم , هتاخد ورقة الأسئلة تعمل بيها ايه , مش هتاخدها كان زمانها اتحلت

> 🧠 ال OS بيشوف ال process محتاجة ايه , لو محتاجة two resources واحد بس متاح والتاني لأ , مش هيديها اللي معاه لأنها مش هتعرف تعمل بيه حاجة لوحده

3) no pre-emption

💢 problem discerption:

الاتنين محدش يقدر يجبرهم انهم يسيبوا حاجة

🍏 solution:

عارف لو المراقب قال ل عصام هات الورقة و اطلع استنى برة , لما حد يخلص ابقى خد قلمه كانت اتحلت

> 🧠 ال OS بياخد ال resources من ال process وبيقولها استني ف ال blocking

💡 خلي بالك ال preemption هنا غير ال preemption بتاع ال scheduling algorithms ,ازاي؟؟

- ال preemption بتاع ال scheduling ده بيحصل على مستوى ال CPU ,يعني باخد ال CPU بس من ال process انما لو معاها أي resource بيفضل معاها
- ال preemption بتاع ال deadlock solution ده بياخد ال resources من ال process كمان

4) circular wait

💢 problem discerption:

من اسمها كدة تحس فيه circle او loop لان كل واحد مستنى التاني

عصام : هات يبني القلم
اياد : هات انت الورقة
عصام : هات يبني القلم
اياد: هات انت الورقة
عصام: هات يبني القلم
اياد : هات انت الورقة

🍏 solution:

✅ بنعمل linear order لل resources لازم ال process تمتلكهم بالترتيب يعني لو ممتلكتش رقم 1 , مش هتمتلك رقم 2

1. قلم (عدد 1 قلم)
2. ورقة (عدد 1 ورقة)

المشكلة مش هتحصل ليه ؟ لأن اللي امتلك القلم الأول هو اللي هيكسب السباق ويمتلك الورقة
مبقاش فيه حاجة اسمها حد يمتلك القلم و حد يمتلك الورقة والاتنين ي circular wait على بعض

deadlock recovery

لو ال deadlock حصل خلاص نعمل ايه ؟؟؟؟؟؟

بنختار process , ال process الغلبانة دي اسمها victim الضحية يعني

بنعملها

1. اما abort عشان تسيب ال resources اللي معاها
2. أو rollback زي بتاع ال database كدة, يعني لو عملنا progress ف ال process بنرجع فيه للحتة قبل ما تاخد ال resource

مش بنختار ال victim process اعتباطيا بيبقى فيه شوية معايير زي مثلا

1. هي بقالها نص ساعة شغالة وفاضلها دقيقة ينفع تقوم عاملها abort وبادئ من الأول فعشان كدة لازم يبقى cost effective
2. ممكن نختارها على حسب ال priority برضو و ف الحالة دي لو كل مرة الحظ وقع على نفس ال process انها تطلع بسبب ان عندها low priority بيحصل حاجة اسمها **starvation** يعني بتموت من جوعها لل resources

---

logical address vs physical address

```c
int sum(int x, int y)
{
        return x + y;
}
```

شايف الكود ده ؟؟؟ تعالى ناخد معاه رحلته من الأول خالص من أيام كود مكتوب بال c

كلنا عارفين ال compiler صح , ال compiler بيحول ال كود اللذيذ ده ل binary code شوية 0 و 1

حلو لحد هنا

محتاجين نقول لل CPU اللي هيعمل execution
يا CPU

- ال instructions بتاعتي هتلاقيه بتبدأ من ال address ده xxxxxxx

> ال address اللي بيبدأ منه ال program اسمه base address وركز عشان مهم

بس فيه مشكلة هنا 💢
مينفعش ال compiler يقول لل CPU ال instructions بتاعتي هتبدأ من العنوان 0x5555 مثلا لان ال compiler ملوش حكم على ال memory , و ميعرفش اذا كان ال address ده فاضي أصلا و لا لأ

طب هنحلها ازاي ؟؟؟؟
ولنفرض اني عندي

Symbol	Size	Address (Logical)	base address
`sum`	50 bytes (approx.)	0	xxxx xxxx
`x`	4 bytes (int)	50	xxxx xxxx
`y`	4 bytes (int)	54	xxxx xxxx

هقول لل OS لما تيجي تعملي memory allocation ايا كان ال base address اللي هتدهولي

- عايز ال sum تبقى بعد ال address ده ب 0 (يعني نفس ال address لانها أول function اخزنها)
- عايز ال x تبقى بعد ال address ده ب50 (عشان اول 50 خزنت فيهم ال sum)
- عايز ال y تبقى بعد ال address ده ب 54 (علشان sum خدت 50 و x خدت 4)

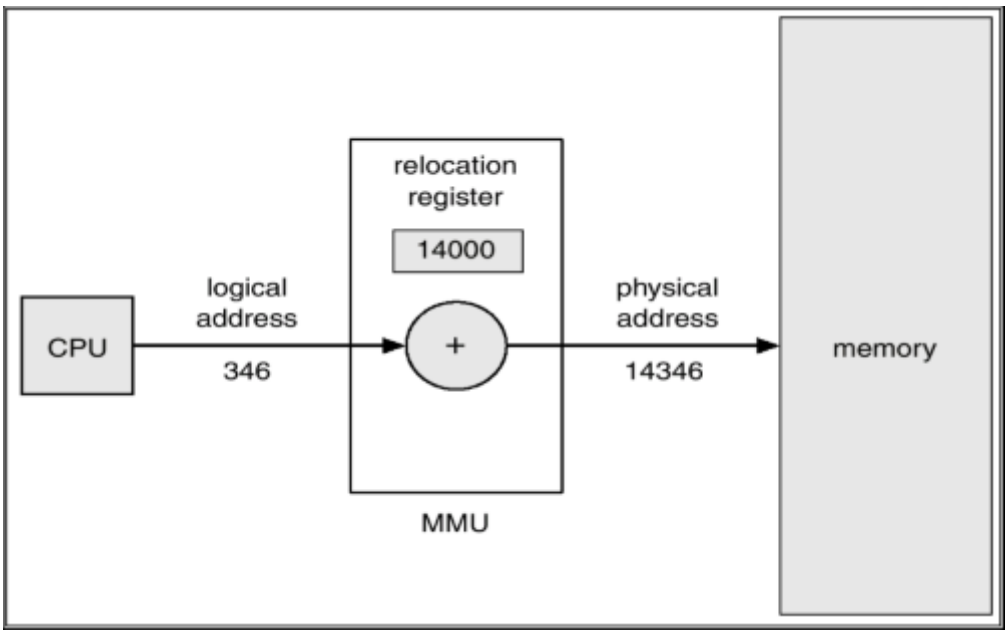
> مش ده اللي بيحصل بالظبط ولكن نفس ال concept

physical address = base address + logical address (offset address)

ال physical address اللي هستخدمه فعليا عند ال execution يساوي

ال base address اللي هتدهوني يا OS + ال logical address اللي هتدهوني ي compiler (ممكن تلاقي CPU بدل compiler على اعتبار ان ال CPU بي fetch من الكود اللي كاتبه ال compiler)

اللي بيعمل عمليه الجمع دي hardware اسمه MMU أو memory manage unit



فاكر ال CPU scheduling ال OS كان عنده algorithms كتير عشان يعرف يدي ال processes المختلفة لل CPU

هنا بقا الmemory allocation ال OS محتاج algorithms برضو عشان يعرف يدي ال processes المختلفة memory يستخدموها

1) contiguous Allocation

1.1) single partition reallocation

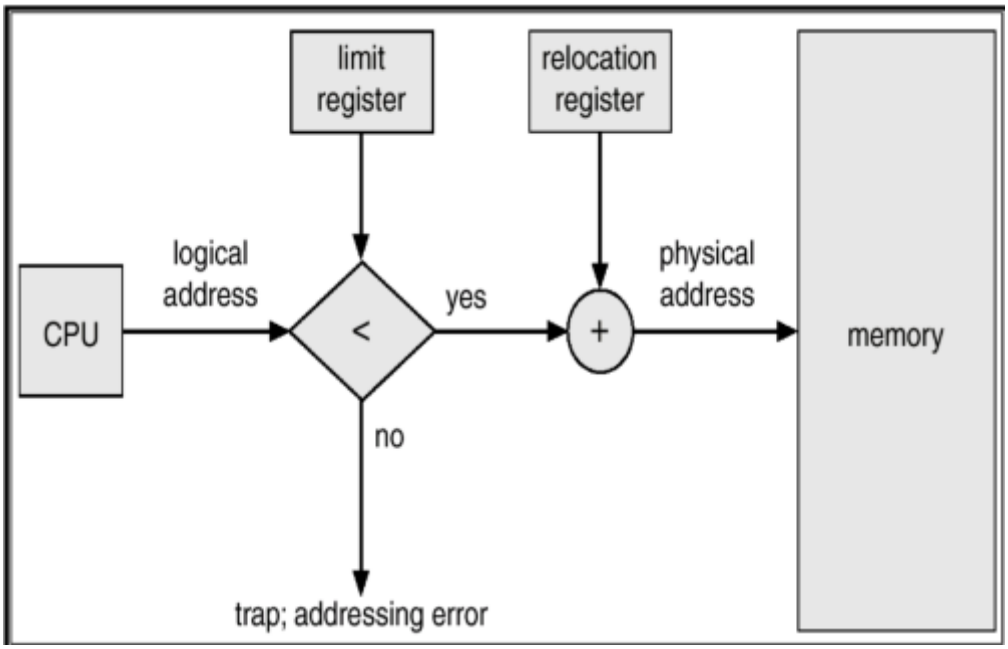
هنا ال memory كلها على بعضها قسمناها ل اتنين partition

- جزء أو partition لل OS
- جزء لل user process

طبعا ده قمة الهبل لأن الجزء بتاع ال user processes مينفعش تحط فيه غير process واحدة بس

اومال عملوه ليه ؟؟؟
ايام زمان بقا قبل التقدم كانوا عايزين يفصلوا ال OS عن ال user process بدل م هم سايحين على بعض ف ال memory فعملوا ال model ده

بيشتغل ازاي؟؟؟
بقى عندنا limit register
ده بنخزن فيه ال limit بتاع ال process دي لو ال process حاولت تستظرف و ت access memory أكبر من ال limit بتاعتها ال limit register بيقفشها و يبعت interrupt لل processor ويبهدل الدنيا

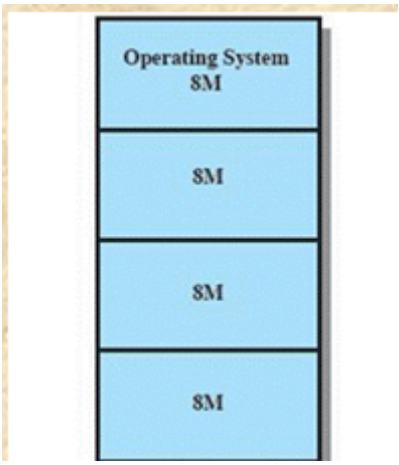


1.2)multiple partition reallocation

1.2.1) fixed partitioning

هنا ابتدينا نقسم ال memory ل أكتر من partition
طبعا ال partition الواحدة مينفعش تحتوى على أكتر من process عشان ال protection

- equal sized partitions

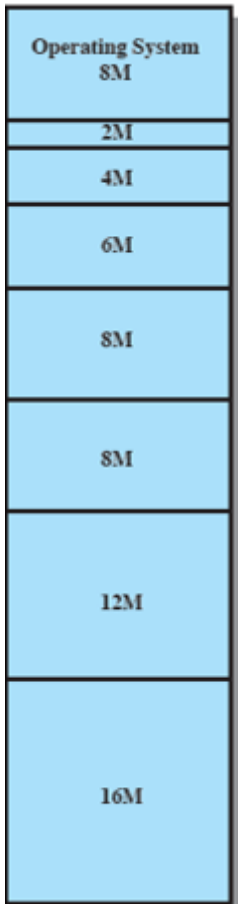


disadvantages:

1. ممكن ال process بتاعتنا تبقى أكبر من 8M
2. لو process ال size بتاعها 1M , هيبقى ضاع علينا 7M بحالهم (عشان زي م قولنا ال partition بياخد process واحدة بس ايا كان حجمها)
   ال 7M دول بنسميهم internal fragmentation

طب ليه internal عشان الفراغ ده جوه المساحة اللي انت حاجزها اللي هي ال partition يعني

- unequal sized partition



- advantages

صلحنا ال disadvantages بتاعت اللي فاتت

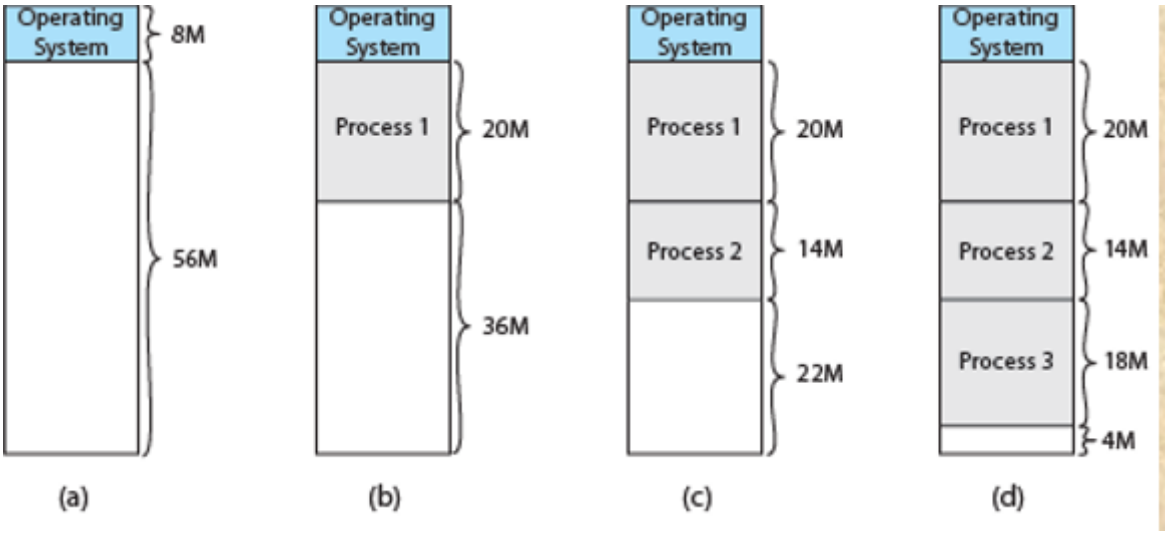
1. بقى عندي size اكبر لل processes الكبيرة
2. لو process صغيرة هحطها ف partition صغير فهقلل ال internal fragmentation

- disadvantages

1. ما زلت مقيد بعدد partitions محدد
2. ال process الصغيرة هتقلل ال internal fragmentation تمام بس هيفضل موجود برضو

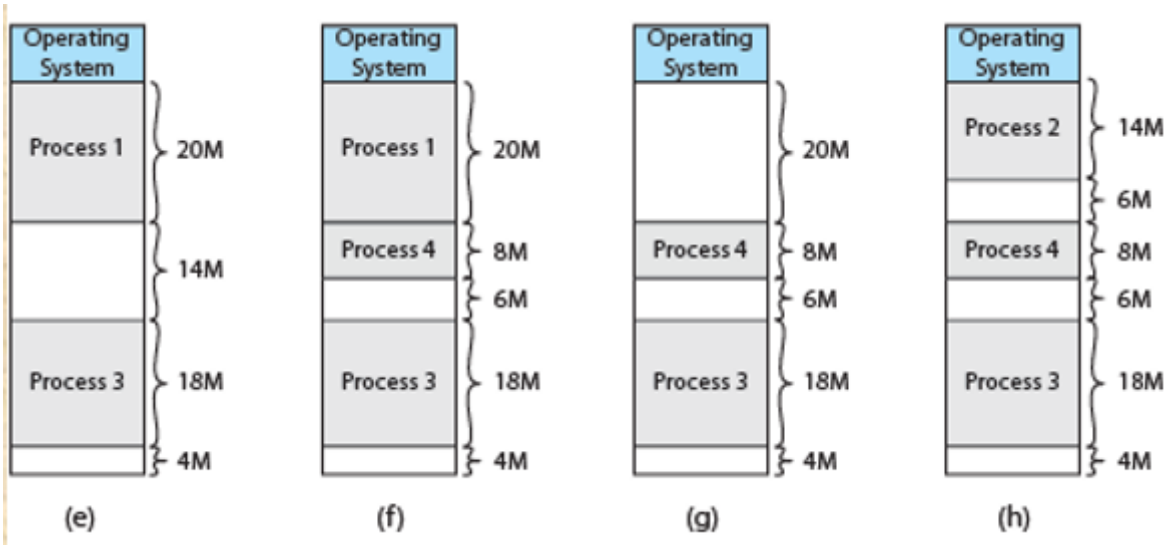
1.2.2) dynamic partitioning

هنا ال OS بيشوف ال process
محتاجة 20M هيحجزلها ال 20M بالظبط
محتاجة 14M هيحجزلها ال 14M بالظبط



ف عشان كدة بيقولك partitions with variable length and number

طب م اهو كويس اهو معقولة ليه disadvantages ؟؟؟؟
ايوة للأسف



ركزلي على ال processes لما بتخلص وتخرج من ال memory
بتسيب external fragmentations (هنا external عشان برة ال partition مش جواه)

طب وايه يعني؟
لو عندنا process ال size بتاعها 16M بالرغم من اننا عندنا 16M فاضيين , الا اننا مش هنعرف نديها لل process عشان ال process محتاجاهم جنب بعض (contiguous يعني) وهم متفرقين زي م انت شايف

طب والحل ؟؟؟
نعمل حاجة اسمها compacting , نضغطهم بحيث المساحات اللي بينهم دي تتجمع على بعض وتبقى block كبير على بعضه

بس للأسف ال processor هو اللي هيعمل ده, وهيأثر على ال performance بالتبعية

placement algorithms

احنا قولنا هو بيعمل ايه بس م قولناش بيشتغل ازاي

1. first fit

من اسمه
بيعمل scan او liner search يعني من اول block ف ال memory , لحد م يقابل أول block مناسب قدامه

2. best fit

ده بيدور على أحسن block بمعنى
لو عايز اعمل allocation ل 5M ومتاح (6M 12M 20M) هيختار ال 6M
مشكلته انه بيعمل scan من اول block ف ال memory لاخر block ف ال memory عشان يلاقيلك أحسن حاجة

3. worst fit

ده عكس ال best
بيعمل scan من اول block ف ال memory لاخر block ف ال memory عشان يلاقيلك أسوأ حاجة (أكبر block متاح يعني) يعني لو محتاج 2M وعندك (6M 12M 20M) هيختار ال 20M

طبعا هو عنده حسن نية انه يجمعلك كذا process صغيرة ف ال block الكبير ده بس ده مش علطول بينفع

2) non-contiguous allocation

2.1) paging

ف النوع اللي فات كان لازم عشان احجز memory لازم تتحجز ورا بعضها contiguous يعني, حتى لو فاكر لما كان عندنا مساحة كافية بس عشان كانت متفرقة على كذا block معرفناش نستفاد منها (وقولنا نعمل compacting)

طب ليه منفكرش اننا ن allocate memory ال process ف بتاعتنا non contiguous blocks ؟؟؟

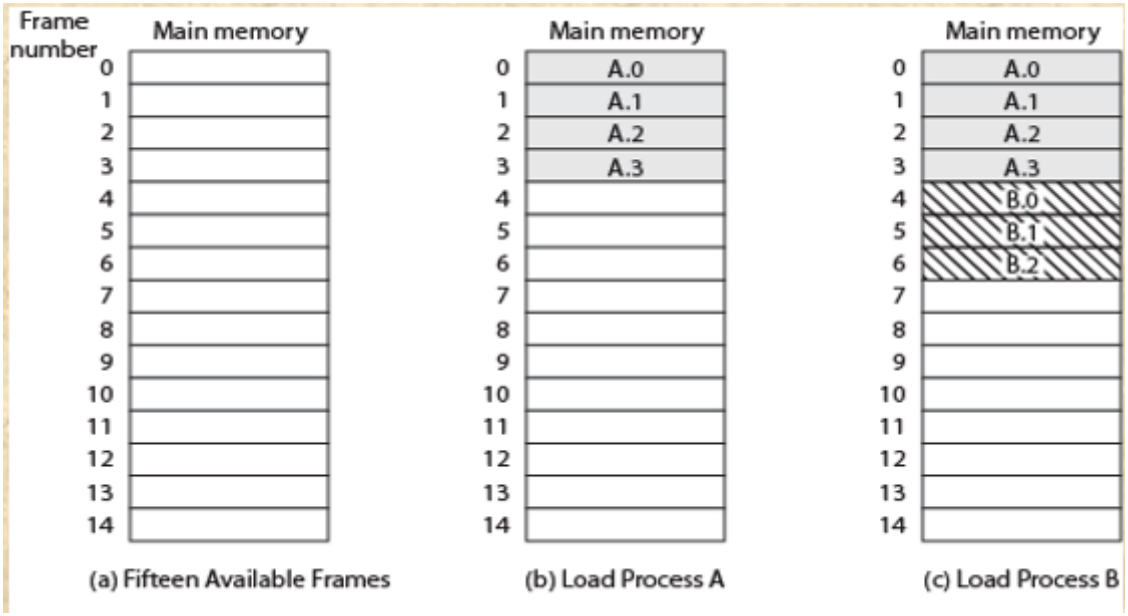
بدل م نقسم ال memory بتاعتنا ل partitions

هنقسم ال memory ل أجزاء اسمها frames
ال frames دي ال size بتاعها ثابت

وهنقسم ال process بتاعتنا ل أجزاء اسمها pages
ال pages دي ال size بتاعها ثابت برضو

> ال size بتاع ال page = ال size بتاع ال frame

ف كدة الprocess اللي ال size بتاعها four pages
محتاجة four frames فاضيين ف الmemory



هتقابلنا مشكلة هنا, محتاج اسجل كل page انا اديتها انهي frame بالظبط ؟؟؟ والا مش هعرف أوصلهم

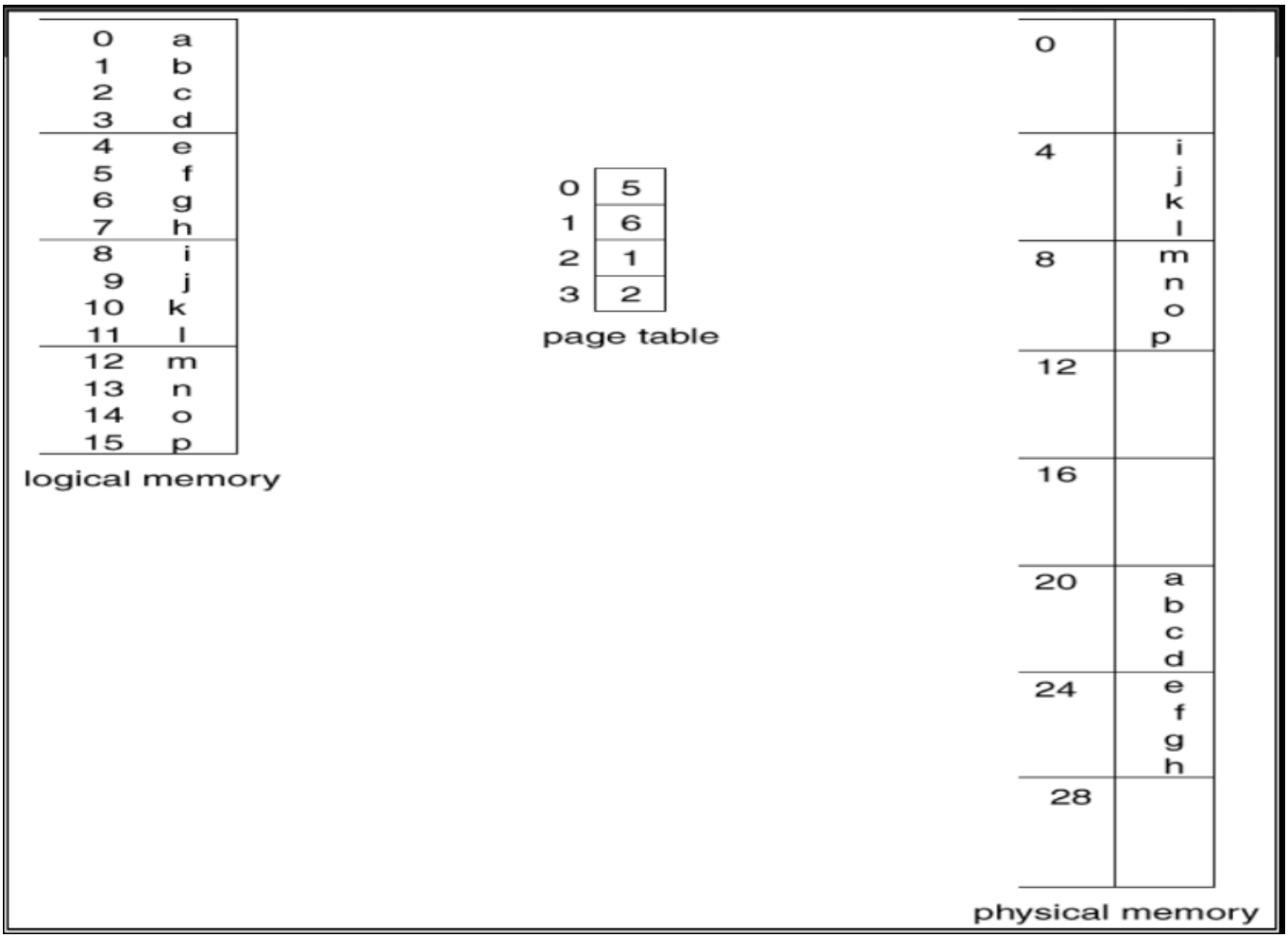
page table

ده جدول ببساطة بيوريك كل page من ال process بتاعتك محطوطة ف انهي frame ف ال memory

Page Number	Frame Number
0	3
1	5
2	8
3	2

ازاي نحول ال logical ل physical:

ف الأول هسألك سؤال لو عندنا كتاب 500 صفحة , وانت برا البيت اخوك كان بيرتب الأوضة و نقلهولك على رف تاني ؟ هتقوله انهي رف ولا هتقوله انت وديت صفحة 50 فين و صفحة 60 فين



مش فاهم؟؟؟

ال page عندنا زي الكتاب بالظبط نقلناها من رف ال (virtual) ل رف ف ال physical
ف عشان اعرف اي byte ف ال page دي فين بالظبط مش محتاج غير معلومة واحدة اللي هي رقم الرف الجديد أو ال base address

عندي logical address قيمته 15 فيه حرف ال p
وانا عارف ان ال page بتاعتي ال size بتاعها 4B (من الرسمة اللي فوق)

يبقى لو عايز اعرف هو ف انهي page ,وبعد ال base address بتاعها ب قد ايه

a) 15 / 4 = 3
b) 15 % 4 = 3

يبقى ال address ده ف ال pageاللي الاندكس بتاعها 3 (عشان بيبدأ من 0 اندكس) و بعد ال base address بتاع ال page ب offset يساوي 3

ف كدة واقفة معايا اني اعرف ال base address بتاع ال page , هروح لل page table اعرف رقم ال frame ومنها اجيب ال base address بتاع ال page لانه بساوي ال base address بتاع ال frame

> ازاي ؟؟؟
> هضرب اندكس ال frame ف ال size بتاعه يعني

لو frame رقم 0 و حجمه 4B
0 *4 = 0 , يبقى ال base address بتاع ال 0 frame يساوي 0*
لو frame رقم 2
2 4 = 8 يبقى ال base address بتاع ال 2 frame يساوي 8

physical address = base address of page + offset = 8 + 3 = 11

هتلاقي فعلا ال p ف logical يساوي 15 بقت ف physical يساوي 11

shared pages

لو فتحت الالة الحاسبة ف 3 ويندوز جنب بعض , كلهم ليهم نفس ال code صح؟؟؟
بدل م اعمله allocate تلات مرات ف ال memory كفاية مرة, احطه ف frame واحدة واخلي ال page بتاعت الكود فيهم كلهم بتشاور على ال frame دي

2.2) segmentation

فالحقيقة
عندنا الكود بتاعنا ده عشان نحطه ف ال ram و نعمله execute مش بيتحط ف حتة واحدة على بعضها , ولكن بيتقسم كذا حتة او segment

- فيه segment بنحط فيها ال instructions أو ال functions بتاعت الكود اسمها ------> code segment
- فيه segment بنحط فيها ال global variables بتاعت الكود اسمها ----------> data segment
- فيه segment بنحط فيها ال local variables بتاعت الكود اسمها ----------> stack segment
- هنقسم ال process بتاعتنا ل segments مش متساوية ف الحجم عكس ال pages.
- و مش هنقسم ال memory ل frame هنسيبها (مش بيفكرك ب ال dynamic فهيبقى فيه external frags)

محتاج اسجل لكل segment

1. base address
   فيها يعني متخزن فين address اول
2. limit(size)
   ده address هي حاجزة قد ايه من أول ال

> مكناش بنسجل ال size ف ال page table لان ال frames كان ال size بتاعها ثابت
> لكن هنا ال segments ال size بتاعها متغير

Segment No	Base Address	Limit
0	1000	500
1	1500	300

segmentation with paging

هنا هنقسم ال segmentation ل pages و بعدين نخزن ال pages دي على frames
ف كدة
كل segment ------> كذا page
كل page----> كذا frame
ده هيقلل ال external fragmentations

Here are the tables without any text:

Segment Table:

Segment No	Base Address	Limit	Page Table Address
0	1000	500	Page Table 0
1	2000	600	Page Table 1

Page Table 0 (for Segment 0 - Code):

Page No	Frame No
0	5
1	8
2	12

Page Table 1 (for Segment 1 - Data):

Page No	Frame No
0	15
1	18
2	22

---

virtual memory

كان بيقولوا ان ال c و ال cpp بتسمحلك تتكلم direct مع ال hardware , عن طريق ال pointers لو فاكر طب ازاي ؟؟؟ اومال ال OS راح فين , و protection ايه اللي بنتكلم فيه بقا

```
int *ptr = 0x3434;
*ptr = 45;
```

ده معناه ان ب كود زي ده كدة انا ممكن ابوظ كل حاجة و ادخل على أي address اشوف ف ايه , أو اغيره صح كدة ؟؟؟؟ لأ

قبل م اقولك لأ ليه تعالى نروح نتمرن عشان انت قاعد تذاكر بقالك كتير بس هنروح جيم تاني المرادي بيقولوا عليه جامد جدا

عصام واياد راحوا الجيم بس اياد اتأخر عن عصام ربع ساعة .....

دخل عصام قام الراجل قاله:
"هات شنطتك ي كوتش, احنا عندنا 100 لوكر اختار أي رقم تحبه من 1 ل 100 , واحنا هنحطلك الشنطة هناك"
عصام اختار 5 واداله الشنطة

دخل اياد قام الراجل قاله:
"هات شنطتك ي كوتش, احنا عندنا 100 لوكر اختار أي رقم تحبه من 1 ل 100 , واحنا هنحطلك الشنطة هناك"
اياد اختار 5 بالصدفة واداله الشنطة

وهم مروحين , اتفاجئوا انهم مختارين نفس رقم اللوكر لكن الراجل ادى كل واحد شنطتة عادي , طب ازاي؟؟؟؟

لاحظنا حاجة تاني غريبة الجيم ممكن يبقى فيه 200 واحد ف نفس الوقت , و كلهم خدوا لوكرز , طب ازاي ؟؟؟ واحنا عندنا 100 لوكر بس ؟؟؟

الجيم دلوقتي بيهندل حاجتين اشبه بالمستحيل

1. اي حد يقدر يختار اي لوكر من ال 100 (مفيش مرة هيتقاله اللوكر ده محجوز اختار غيره)
2. كل الناس هتلاقي لوكر متاح ف أي وقت حتى لو الجيم فيه ناس أكتر من عدد اللوكرز

طب ده ازاي ؟؟؟

1. الجيم مفهمك انه عنده 100 لوكر
   💢 بس اللي متعرفهوش ان عنده مخزن فيه 1000 لوكر (بس بعيد بحوالي 500 متر)
2. الجيم بيستخدم **جدول**

سيناريوهين تعالى نشوف بيتهندلوا ازاي:

1. ازاي اي حد يقدر يختار اي لوكر من ال 100 (مفيش مرة هيتقاله اللوكر ده محجوز اختار غيره)

هياخد منك رقم اللوكر اللي تحبه , ويحطه ف المكان اللي هو عايزه مش اللي انت عايزه (بيشتغلك يعني) ويكتب الكلام ده ف الجدول , ولما تيجي تطلب حاجتك , هيبص ف الجدول يشوف هو خزنهالك فين فعليا ويديهالك وانت غلبان هتنبهر

> هنا A اختار 8 بس هو حطها ف 20
> و B اختار 8 برضو بس هو حطها ف 30

name	num of locker he picked	num of actual locker
A	8	20
B	8	30
C	9	1

2. ازاي الناس اللي ف الجيم أكتر من عدد اللوكرز , أكتر من 100 يعني

name	num of locker he picked	num of actual locker
A	8	20
B	8	30
C	12	storage (المخزن)

هيملى ال 100 لوكر بعدين , بعدين يبدأ يحط ف المخزن 🔥

المخزن 🔥🧨😱
صدقني صاحب الجيم مش هيحب يروح المخزن خالص , عارف لو بص ف الجدول ولقى شنطتك ف المخزن بيعمل ايه؟؟؟؟؟

1. هيختار لوكر من ال 100 ويطلع الشنطة اللي جواه
2. بيروح المخزن 500 متر يجيب شنطتك من هناك و يحط مكانها الشنطة اللي طلعها من اللوكر (swapping)
3. هيعمل update ف الجدول , الشنطة اللي انت خدت مكانها بقت ف المخزن و شنطتك بقت ف لوكر
4. بيطلع الشنطة من اللوكر قدامك وانت تنبهر

ف كدة C لو هياخد شنطته اللي كانت ف المخزن
الجدول هيبقى كدة

name	num of locker he picked	num of actual locker
A	8	20
B (الشنطة اللي انت خدت مكانها)	8	storage

name	num of locker he picked	num of actual locker
C	12	30

حصل swapping بين b, c g لو واخد بالك

خلصنا الجيم تعالى نشوف بقا ال virtual memory

الجيم ------> الOS
اللي داخل يتمرن -----> ال process
ال 100 لوكر المتاحين طول الوقت تختار منهم -----> virtual memory
ال 100 لوكر اللي جوا الجيم فعليا ----> ram
المخزن ---------> hard disk

اي process عندنا , ال OS بيقولها ال ram كلها بتاعتك لوحدك اختاري أي address space تحبيها
ال process بتقوله خلاص انا عايزة ال address 0x3434 احط فيه variable X

ال OS بيقولها تحت أمرك طبعا, و ياخد ال X يحطها ف حتة على مزاجه وليكن 0x2222
وبيسجل عنده ف ال table

variable	virtual address	physical address
x	0x3434	0x2222

طيب مفيش مكان ف ال ram خلاص ؟؟؟؟
خلاص هحطها ف ال disk و اسجل ف الجدول عندي انها ف ال disk

variable	virtual address	physical address
y	0x5912	disk

فيه ملحوظة صغنونة
لو عندنا ram 4GB يعني 1 مليار address 😱 (ال address الواحد بيلم 4 byte تحته)
لو سجلناهم ف جدول محتاجين 1GB للجدول بس وده مينفعش أكيد

فعشان كدة هنقسم ال virtual memory ل pages وال physical memory ل pages ويبقى الجدول كدة

page virtual address	page physical address
0xe234	0x3000

الpage -----> الشنطة

page demanding and page fault

طب ال process طلبت تقرا من address وال OS بص ف الجدول لقاها ف ال disk

ال DISK 😱🧨
دي مصيبة كبيرة عشان اروح اجيب حاجة من ال disk , لو فاكر المخزن 500 متر يعني هيبقى بطيئة خالص

السيناريو اللي بتضطر اروح اجيب فيه حاجة كنت مخزنها عال disk اسمه **page fault**

1. ال OS بيختار مكان ف ال ram (مش عشوائي) يطلع ال page اللي فيه ويبص فيها

- لو dirty page يعني حصل فيها تغيير عن النسخة اللي عندي على ال disk , لازم احفظ التغييرات دي على ال hard disk الأول قبل م اشيلها من الram
- لو clean page يعني متغيرتش عن النسخة اللي عندي على ال disk , يبقى هشيلها من الram وخلاص (م كدة كدة هي متغيرتش ومعايا النسخة هي هي ف ال disk)

2. ال OS بياخد ال page اللي الprocess طلبتها, وينقلها من ال disk لل مكان اللي احنا فضيناه ف ال ram
3. ال OS هيعمل update لل page table , ويقول ال page دي مكانها اتغير بقى ف ال ram والpage دي مكانها بقى ف ال disk
4. هتقرا ال data اللي كنا عايزينها من ال ram عادي خالص , محدش شاف حاجة

demand paging & lazy swapper

فاكر مشوار المخزن البعيد ده , احنا قولنا انه مشوار مش حلو خالص ف لازم صاحب الجيم يقلله على قد م يقدر
لانه بيكسل 😑 تروح تقوله هات شنطتي , يقولك انت لعبت كتف جانبي ؟؟؟ تقوله يعم انا خلصت , يقولك طب انزل 4 * 10 ضغط وتعالى هجيبهالك

بالمثل لازم اقلل المشوار لل disk مش اروح واجي كل شوية
بمعنى لو عندي program عبارة عن 50 page هل هحتاجهم كلهم ف نفس الوقت اكيد لأ اعمل load للي page اللي انا محتاجها بس دلوقتي (demand page)
هنا ال lazy swapper هو ال os مش هيجيبلك ال page غير لما يطلع عينك

page replacement algorithms

عايزين نفضي مكان ف ال ram لل page اللي جاية من ال disk
نختار أي page ف ال ram دلوقتي ونرميها ف ال disk وخلاص؟,
لأ, الموضوع مش عشوائي زي م قولنا قبل كدة

1. FIFO(first in first out)
   اقدم page هتطلع برة FIFO) عشان هي اول واحدة دخلت يعني ف يبقى اول واحدة تطلع(
2. optimal
   من اسمه يبان جامد بس تنفيذه صعب , بيقولك هطلع ال page اللي مش هنستخدمها ف المستقبل (طب هتعرف منين)
3. LRU (least recently used)
   هنا مش اقدم واحدة ف ال memory لا أقدم واحدة حصلها استخدام

virtual memory advantages:

1. memory protection
2. support for large address space (اللوكرز اللي مش بتخلص ف الجيم)

كدة خلصنا , تقدر تجاوب على السؤال اللي سألناه ف الأول خالص

- عشان ال OS بيقول لل program: قدامك ال memory كلها , اختار براحتك يا حبيبي, اختار
- بيختار address 0x3434 , غلبان بقا فاكر انه يقدر يشوف ال address فعلا لكنه مضحوك عليه زي م شوفنا

file management

file allocation

المشكلة؟؟؟؟

عايزين نعمل file و نحطه ف ال disk (نعمله disk allocation يعني)

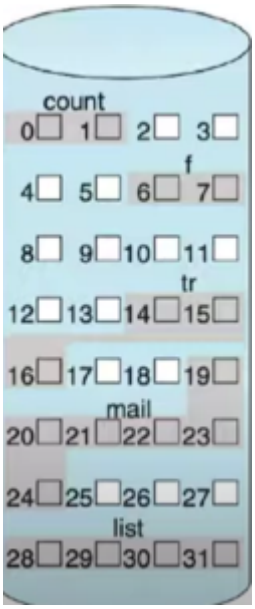
بتشبه ولا ايه , ايوة انت صح , ده شبه سؤال عندنا process وعايزين نحطها ف ال ram , ف لو مقرأتش ال chapter بتاع ال memory management اقراه وتعالى كمل

ملحوظة

احنا بنقسم ال ram ل frames

و بنقسم ال disk ل blocks (حتت صغيرة يعني 4KB مثلا)

هوريهالك برضو عشان تبقى متخيل (المربعات الصغيرة المترقمة دي)



ملحوظة أخيرة لو انت من اللي كان بيتقالهم ان ال directory هو ال folder (كلام مش دقيق)

ال directory هو حتة جدول بنخزن فيه

كل file name

قدامه

كل بياناته ومكانه على ال disk او بيسموها (file control block)

file name	file control block
name	meta data, file location on disk

1) contiguous allocation

بيقولك لو هتخزن file على ال disk لازم تخزنه ف contiguous blocks (جنب بعضها يعني)

بص مثلا على الرسمة اللي فوق , هتلاقي file اسمه count واخد ال blocks رقم 0 و 1 اللي جنب بعض

ف ال directory بتاعي هيبقى شكله كدة (اختصرنا وكتبنا ال location بس)

file name	start	length
count	0	2

طبعا مش محتاج اقولك اني مش محتاج اخزن غير ال start و ال length بس عشان هم contiguous زي م انت عارف (بيبدأ من ال block رقم 0 وواخد 2 , يعني 0 و 1)

العيوب ؟؟؟؟

لو حذفنا file هيسيب مكانه فراغ (external fragmentations) وطبعا عشان احنا هنا contiguous ممكن يبقى عندي file محتاج four blocks و ال four دول متاحين عندي بس متفرقين فمش هستفاد بيهم 🗑️

الحل ؟؟؟
نزق ال blocks عشان تبقى جنب بعضها مفيش بينها فراغ فيما يعرف ب ال compacting

> محدش بقا بيستخدمها دلوقتي كدة كدة

file growing

مش سهل ت extend ال size بتاع ال file لان ممكن ال block اللي بعديك تكون محجوزة

2) linked allocation

المشكلة ف ال technique اللي فات اني لازم الاقي contiguous blocks

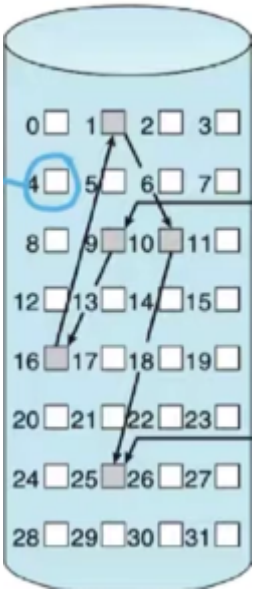
هنا بيقولك مش شرط contiguous , ممكن تخزن ال file بتاعك ف blocks مش جنب بعض (عادي)

فاكر ال linked list ؟ كانت ال node عبارة عن ايه؟؟؟ data و pointer
كان ايه أهم حاجة تخزنها عنها ؟؟؟ ال head بالظبط , ولو تخزن ال tail كمان تبقى برنس

احنا هنعامل ال block ك node هنحط فيه data بتاعت ال file و pointer لل block اللي بعديها

```
1st block(head) -> 2nd block -> 3rd block -> 4th block (tail)
```

file name	start (head pointer)	end (tail pointer)
jeep	9	25



ف الجدول اديتك ال head اللي هو ال block 9
بص على الرسمة كدا

25 <------ 10 <----- 1 <----- 16 <---- 9

المميزات

1. مفيش external fragmentations و لو عندي block متنطورين هستفيد منهم عادي
2. سهل اني extend ال size عشان هعمل block واشاور عليه من ال tail

العيوب

1. ال pointer اللي ف اخر كل block ده مش بياخد size برضو

3) indexed allocation

ده شبه اللي فات بس بيختلف ف حاجة

بدل م ال pointers كانت متنطورة(متفرقة) ف ال blocks بتاعت ال file , جمعنا ال pointers دي ف block واحد اسمه ال index block

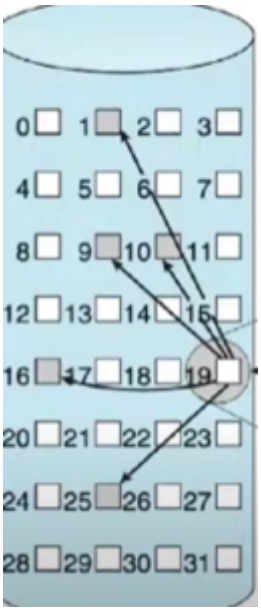
شكله عامل كدة

معناه ايه؟؟

1st block of file ---------> is block 9
2nd block of file ---------> is block 16
3rd block of file ---------> is block 1
4th block of file ---------> is block 10
5th block of file ---------> is block 25
6th block of file ---------> is block -1 -----> indicate end of file

filename	index block
jeep	19

ال block رقم 19 ده هو اللي هلاقي فيه ال pointers بتاعت ال blocks بتاعت ال file



free space management

المشكلة؟؟؟

بعد م ال files يحصلها delete من ال disk

عايزين ن reuse المساحة دي بأفضل طريقة

محتاجين نعرف ال blocks ال free اللي عندنا , ازاي؟؟؟؟

1. bitmap

هيبقى عندنا bitmap حاجة عاملة كدة (10010010101)

كل bit بت map ل block بحيث ال free = 1 , used = 0

2. linked list

هنعمل linked list من ال free blocks

لما نحب ن allocate block هنعمله removal من ال linked list عادي خالص

3. grouping

هنحط pointers لل free blocks ف block واحد)شبه ال indexed allocation كدة(