

Chapter (4)

* علشان التخلص من Process و Thread في العمودي و ينفذ جزء منها بطريقة Sequentially فلو عندي Process ينفذ هنا **Single Port** واحد بس في الوقت و يخلي اخر بقى له السوا **threaded Process**

الآن لو نفذت اكتر من Port في Process II نحن no Port no Parallel بعدها **Multi Threaded Process** بدل من **Parallel** Time is no Slice كل Thread ينادي Single processor ولو عندي Processor隻 Thread متعدد **Multi Processor** وكل Thread يبيسفيه **Code** من ضمن الـ Thread

* كل لها اعمل **Create** ده بياخد مساحة في العمودي heavy weight ← Process Creation بيسفيه الـ Resources Multi Create يذكر لو انا عندي one Process و بـ **One Creation** فتكده هو هستقيد من كل الـ threads بالـ **Creation** threads و باليك الـ Process فيسيفيها light weight creation

هذا عالم word II مثلاً **Multiple threads** في نفس الوقت البرنامج **Spelling Check** و **grammer check** ده في واحد Process ده في نفس الوقت و كل ده في **one word**

Thread دلـ id بيكون معرف لا OS كل Thread * thread دلـ id بيكون معرف لا OS من خلال OS stack space و Register Set و PC كل Thread دلـ id بيكون معرف لا OS من خلال OS thread دلـ id بيكون معرف في الـ **Process** ده يذكر في حاجات **Resources**, **Code**, **data** II دلـ idThreads دلـ id files II دلـ id

Code	data	files
Register	PC	Stack
} thread		
Single thread		

Code	data	files
Reg	reg	reg
PC	PC	PC
Stack	Stack	Stack
} } }		
Multi Thread		

Multithreaded Server || **Multi-threaded** || **آخر** *
وهو Web Server || Request comes from Clients || **Architecture**
will handle heavy requests as new process does request ||
Request || Client process || G new thread does request
light requests on Client

• Multithread II vs. Benefits II *

Thread 8 يسمى execution time أو يكون مستقر حتى لو execution time أو Responsiveness
block يعني دخول في ال wait وهو لأن في threads تابع لـ threads
ينتظر لا يكرر لو كانت تشغيل single فيه حاله دخول ال process
كلها تدخل في ال process wait داخل تابعه و
بالنهاي ال user interface ال response time هو تابع لـ Multi

Threads ١١ Shared Resource ١١ ١١ & Resource Sharing ③
و هنئي دحتاج Share (Chapte١٣) لما كنت بعوز أ
 حاجة بيه ال Shared memory (Processes) فهنا هيفتنى دحتاج
 الكلام ده وبالتالي وخرت الوقت الى كنت بيعمل فيه الكلام ده

Process Cost less than Economy of overhead thread switching creation overhead and context switch cost was less than Process time and it was held until the process needs data or code and then it moves to another process for the second time and it waits for the needs of the threads to move to the main thread.

Multi processing و Multi thread هي بيكون هي Calability ③
و Thread هى Efficiency حيث هتقدر اكتثر هى تفعى
الوقت فالدري هى تكون اسرع

Programmers || دبليو دبليو دبليو || Multi Core Programming (ساعي دبليو دبليو دبليو) *

Dividing activities ①
يقتسم العمل إلى جزءين بحسب Thread Task

balance ②
يحيط قادر Thread Task من Fun + Task

Data splitting ③
يسplit fun إلى Data Task

Data dependency ④
Tasks Task يحيط synchronization Task

Testing & debugging ⑤
التحقق من خطا Task

Concurrency & Parallelism
الفرق بين Parallelism و Concurrency
الـ Parallelism يحيط execution Task في نفس الوقت
الـ Concurrency يحيط execution Task في كل واحده متعدد

Core 1	T1	T3	T1	T3	T1
--------	----	----	----	----	----

T1	T2	T3	T4	T1	T2	T3	T4
----	----	----	----	----	----	----	----

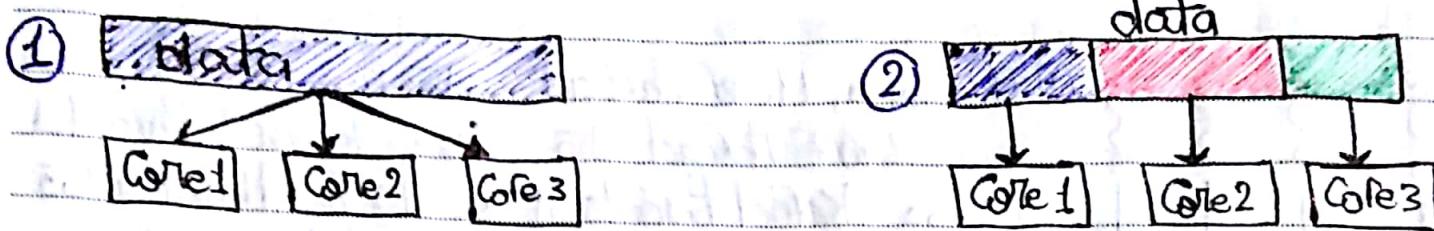
Core 2	T2	T2	T2	T4	T2
--------	----	----	----	----	----

Concurrency

Parallelism

* أنا عندي نوعين من ال Parallelism Thread و فيه يقسم ال Cores إلى Threads حيث كل Thread يعمل operation غير المتاثر . النوع الثاني هو data و فيه يقسم ال data إلى أجزاء و ا minden ال Data Parallelism .

ال Thread يقسم ال data إلى مجموعات cores و يتطلب كل cores إنجاز operation . operation يجري في كل cores و يطلب كل cores جزء من ال data .



* أنا بكتب ال Creation Process ال بتابع ال Code ال يجعل Process Manage User threads يعني ال threads ال threads ال thread library دو هو ال threads Kernel ← Create دو بت التعامل مع ال threads دو بت Schedule و هم دو ال التي هي تعاملها على شكل threads والي بيـ threads management .

POSIX threads ← User threads

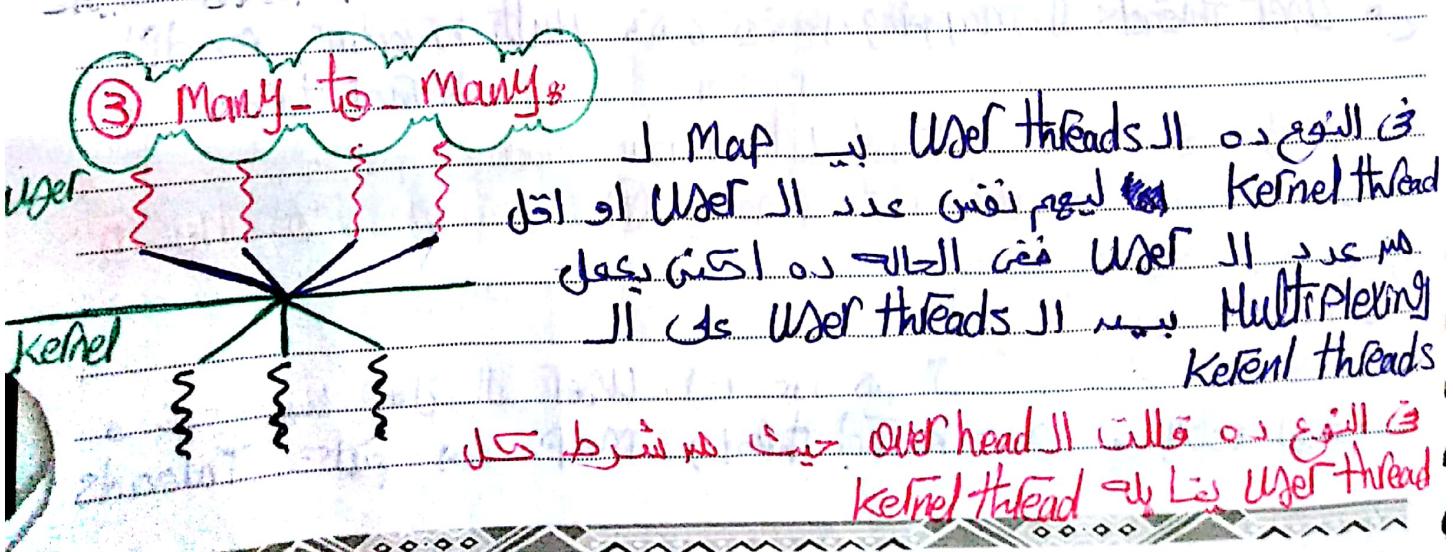
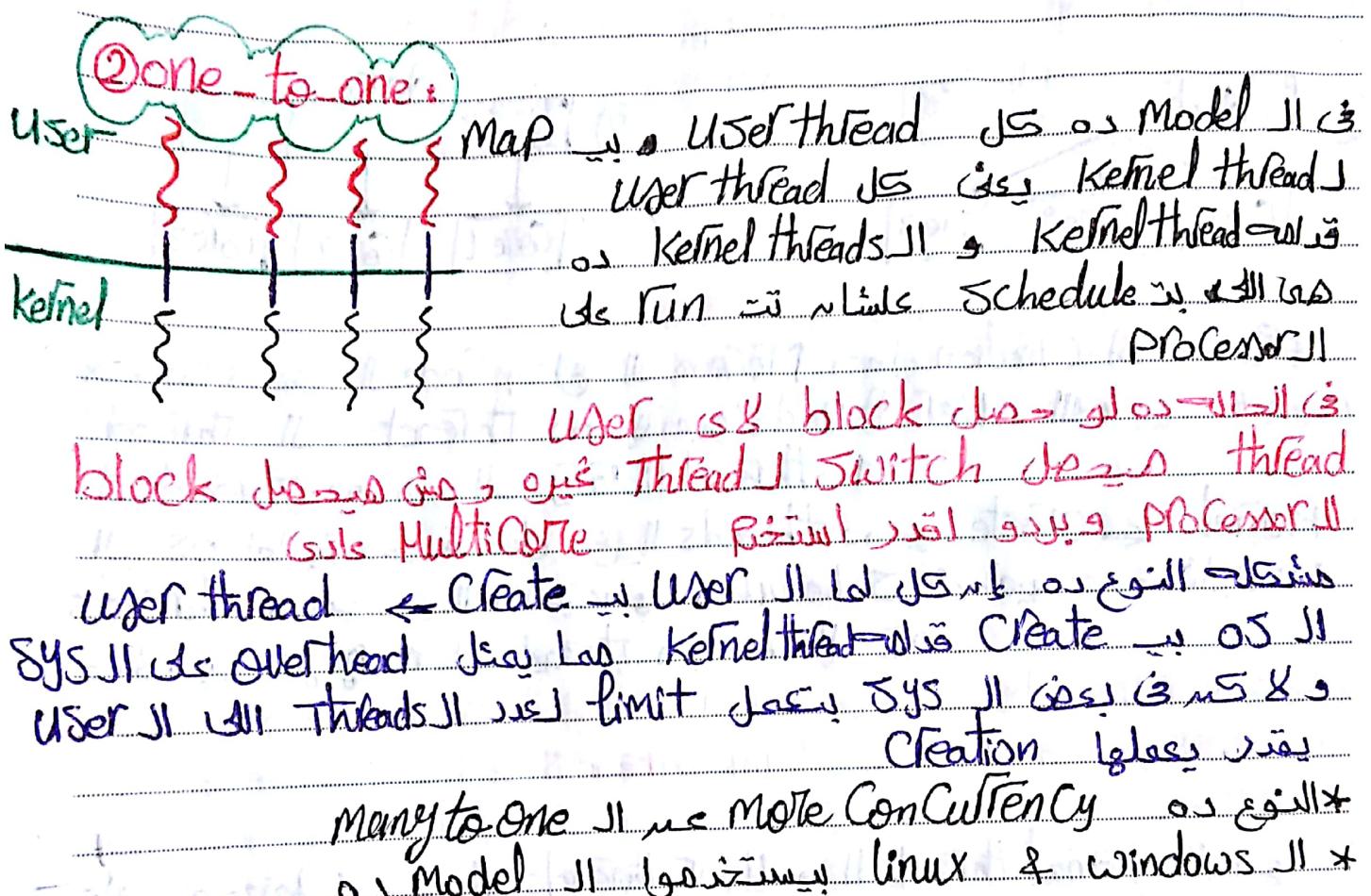
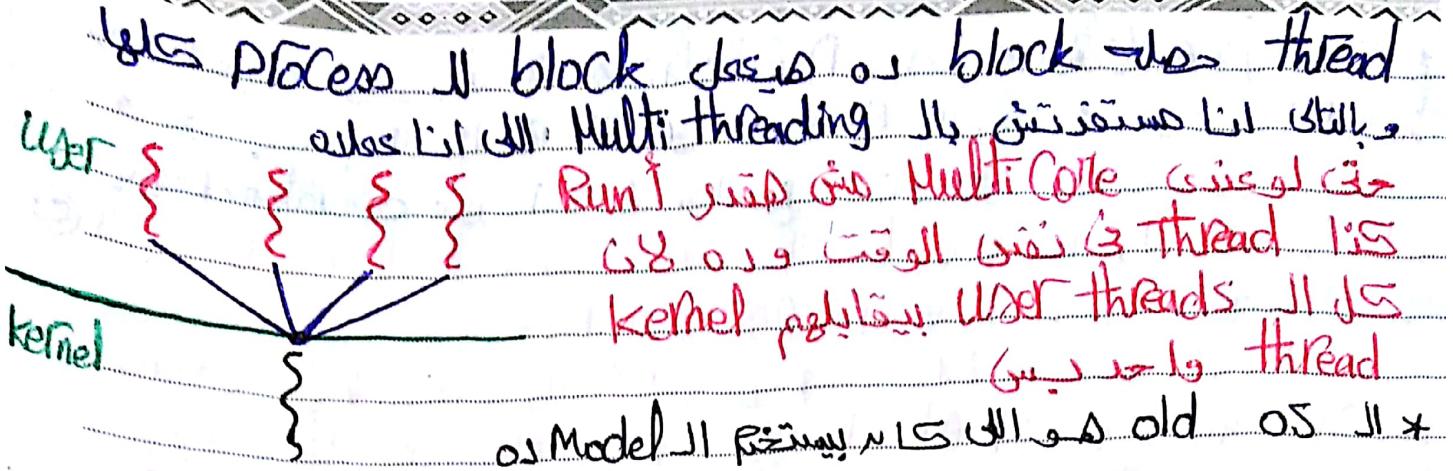
Windows threads - Java threads

لما Kernel threads و User threads دو ال بـ Relation many-to-one مع User threads ال Mapping دو هي Multithreading model

many-to-one ① & Multithreading Model ② many-to-many ③ one-to-one ④

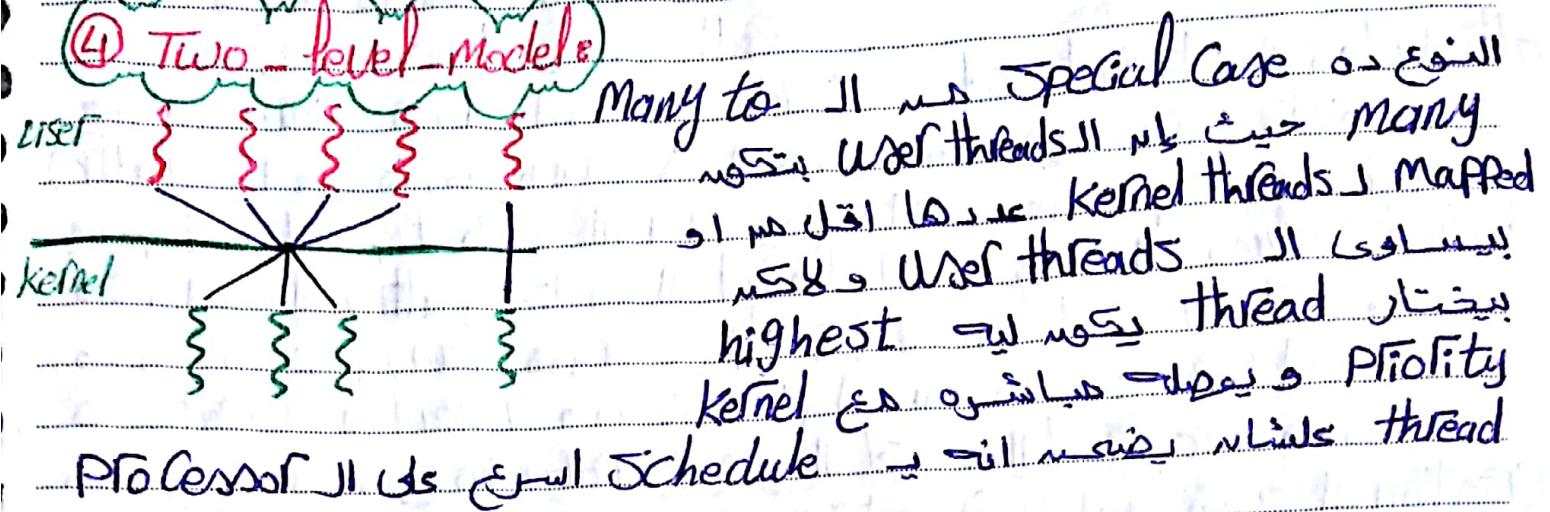
① Many-to-one

و فيه دو User threads اي عدد محدود و one kernel thread map كلهم threads



النوع ده لو جدول
one thread one task | block |
Multi Core | المتعدد في الـ

(4) Two Level Model



في الـ Linux يـ thread قارن في الـ انتخاب ده

Thread Libraries

User thread الـ Manage بـ library
يـ include في الكود بـ API التي اقدر
استخدـها كان Thread Management & Create
Thread Linux على لـ Pthread الـ include class
Windows على l.lib
library الـ fun ده API
implemented in ده API بـ يقول ان الـ API
يـ implementation ده User Space محتاج
implemented in kernel Mode API الـ OS SYS Call
implementation وده ده طريقة الـ

user level API و فيها بعض الـ Pthreads
بعض في الـ POSIX و مستخدم في الـ kernel level
الكلام ده دا وزا اكتب بـ C بـ جمـع اـقام مع
باقي N حيث N رقم درجة الـ User و البروتوكـل ده عـاد اـشتراك
$$\sum_{i=0}^N i$$
 Multi threads

لدي كده الـ runner function اتحسب فيه ss اتحسب في الـ Sum الـ join سيعمل return
فهذا هو المقصود بالـ join يعني هنا join main thread will return
انا اساوي اتفعل join thread will main main thread will return
Thread-join() will wait done will sum will return
يطلع (tid) parent thread will join child thread will return
الـ join will join child thread will return main thread

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>

int Sum;
Void *runner(Void *Param);
```

```
int main (int argc, Char *argv [])
```

5

pthread_t tid;

pthread_attr_t attr;

`pthRead_attr_init(&attr);`

```
pthread_create(&tid, &attr, runner, argv[1]);
```

pthread_join(&tid, NULL);

```
printf("Sum = %d", sum); }
```

Dar kareem

Slides P. 20

~ Function Functions

Void * runner (Void * param)

{ int i , upper = atoi (Param);

Sum = 0;

for (i = 1 ; i <= upper ; i ++) { Sum += i; }

pthread_exit (0);

}

int main () {
 int sum = 0;
 for (i = 1 ; i <= upper ; i ++) { sum += i; }
 pthread_exit (0);
}

الـ sum يخزن في متغير sum بعد كل loop ، upper - هي Variable هي
 معرفة في main .
 main() يرجع بـ pthread_exit () لـ API pthread .
 main() يرجع بـ execution () .

في الحال دو واحد دب قدر عدد threads .

و عدد threads .
 threads .

define Num_Threads = 10

Pthread_t workers [Num_Threads];

for (int i = 0 ; i < Num_Threads ; i ++)

Pthread_Join (workers [i] , NULL);

P.22
Slide

(Num_Threads) و workers .

Scheduler activations

Many to Many Model

User threads .
 ينبع من Kernel thread .
 عدد threads .
 طبقاً للـ SS .
 Creation .
 OS .
 بتحقظ .
 بحاجة .
 و هو .
 light weight process .
 light weight process .
 virtual processor .
 حيث أنها تختار على User thread .
 one kernel thread .
 physical processor .
 thread .
 processor .

طب دلوقت أنا عارف عدد الـ light weight process .

data structure

kernel threads II وظائف kernel threads
حيث UpCalls لغافر يدخل في عدد الـ lightwei threads
thread library وهو موجود في الـ UpCall handler
خلال execution of UpCall class OS يحدد عدد
Creation لغافر لـ User threads

Windows threads

نماذج Windows threads
التشغيل فيه اثنين درجه مسح يجري في الامتحان
Kernel (3 منفذ بمحض API) وال one-to-one *
level

Private data & Stack, Registers, id يعطى thread ID *
لـ dynamic & run-time libraries II وظائف storage area
User Mode II (3 منفذ لـ User Stack) و Kernel Mode II (3 منفذ Kernel Stack)

Context ← Private storage area, Stack, Register set II سبب *
of the thread

و windows II يخزنها في الـ data structures II *
(kernel thread) KThread, (executive thread block) ETThread
(Thread environment block) TEB, (block)

و Thread Start address (3 منفذ ETThread ID) هو المدخل
KThread II Pointer, Parent Process II Pointer
Scheduling & synchronization information II (3 منفذ KThread II (3
TEB II Pointer, Kernel Stack (3 منفذ
Thread, User Stack, Thread ID II (3 منفذ TEB II (3
User Space II (3 منفذ TEB II (3 Local Storage
kernel space II يحتوي على KThread & ETThread II *

و ① في slide 3 (28) ← is linux thread II *
Report II في امتحان لا يوجد في Linux





Windows Threads Data Structures

