

سوال ۱
(الف)

user level thread	kernel level thread
پیاده سازی آسان توسط کاربر	پیاده سازی پیچیده توسط OS

Context switch زمان آوردن حافظه مشترک	Context switch زمان آوردن حافظه مجزا
بدون نیاز به سخت افزار می توان Context switch انجام داد	نیاز به سخت افزار دارد

اگر process block شود تمام process متوقف خواهند شد	اگر thread block شود، بقیه thread ها به کار ادامه می دهند
(ب) ورود داده های اجرایی منتهی می شود از هر سرین تفاوت آنها این است که thread های یک process	

همه shared memory دارند اما در process از حافظه های مستقلی استفاده می کنند

می توان گفت که thread زیر چرخش process هم می تواند باشد. اگر process اجرا می شود، thread را می توان به صورت موازی دانست.

در thread زبان ساخته شدن منابع: stack, PC, Thread ID و مجموعه ای از رجیسترها دارد.
data, file, code را به اشتراک می گذارد.
اما در process یک PCB بر آن اتصالاتی می دم شامل open files, memory map, list variables است که اینها مختص داده thread ها نیستند و خواص پردازشگر.

(۱) switching در thread ها (در یک process) فضای واقعی میانی دارند و به همین

دلیل مقدار cache و memory address space نمی‌تواند اساساً process switching باشد

PCB قبل از خود در PCB بعد را load کنیم، مقدار cache و TLB هم reset flush می‌شوند زیرا

memory address space در آنها متفاوت است، هر چند این کار با زیاد است و در نتیجه زمان بیشتری می‌برد

Year:..... Month:..... Day:.....

(۲) اگر کسی عریف با فرمان قائق مت راست غور را بردارند، به حالت Hungry

می‌آیند و بعد متراً قائق مت می‌شوند و به دلیل نبودن آن می‌توانند غذا نخورند و در شروع کنند

در همین حالت به صورت waiting باقی خواهند ماند.

سوال ۲) در اینجا از `fork` برای ایجاد `child process` استفاده می‌کنیم. به همین دلیل زمانی که

به `child process` می‌رسیم `pid = 0` باشد مقدار `value` در آن ۰ می‌گردد، اما چون می‌بینی از `value` ایجاد کرده‌ایست، با تغییر

`pid > 0` مقدار `value` در `parent` را چاپ می‌کنند همچنان که است.

اگر می‌خواهیم مقدار آن ۲۰ شود از `fork` باید استفاده می‌کنیم (اما الان می‌خواهیم از `process` ما در `variable` های
صفحات عملیات انجام می‌دهند.

۳) مداره خواهر بود (در حقیقت) ترها با تعدادی زمان بتوانند کار خود را انجام دهند و به دلیل نه اینکه ما گاهی
 برای اجزای توانیم در تقارن یک $thread$ های A و B به صورت یکی در میان به مقدار n اجرا فرمایند.
 همچنین با توجه به اینکه در کور اجرا کردن برای هر کدام حداقل برابر انجام خواهد شد، کمترین مقدار ظرفیت تواند
 باشد. همچنین چون مقدار ظرفیت را از یک شتر است، حداقل به $n = 2$ می آیم. برای اثبات این

Year:..... Month:..... Day:.....

ارای سوال ۳) موضوع آخرین $1 +$ را در نظر بگیرید. قبل از دادن حداقل یک بار این عمل توک خود تر را
 نزدیک مقابل انجام شده است، پس مقدار $n = 1$ است و با انجام آخرین مرحله، $n = 2$ می شود.
 یک حالت برای $n = 2$: تر A و B فرمان حلقی اول را شروع کنند، تر A به حلقی ۱ برسد و بلافاصله
 به $store$ آن، حلقی اول تر B آغاز می شود، $n = 2$ پس حلقی دوم B و نیم A فرمان شروع شوند
 و با اتمام حلقی ۵ در B ، حلقی ۵ A هم مقدار n را به 2 می نواز (آخرین تر).

A	B
load 1, inc 1, store 1	load 1, store 1, inc 1
loop 2, loop 3, loop 4	inc 1
load 5	store 1, $\rightarrow n = 1$
inc 5	loop 2, loop 3, loop 4, loop 5
store 5 $\rightarrow n = 2$	

همچنین تمام مقادیر بین 2 تا 10 هم برای n ممکن هستند (در شمار $loop$ های این تر که کمتر از مقدار n شتر می شود)

(۴) طبق law of diminishing returns: $speed\ up \leq \frac{1}{s + \frac{(1-s)}{N}}$
 که در آن s = تعداد سرورهای موجود و N = تعداد سرورهای مورد نیاز

طبق این قانون اگرچه بهر چه با افزایش تعداد سرورهای موجود، افزایش سرعت خواهد بود، زیرا این به سرعت
 واحد و در هر چه این بخش سرورهای بزرگ باشد، مقدار زیادی از این سرعت با زیاد کردن سرورهای کوچکتر و در هر چه

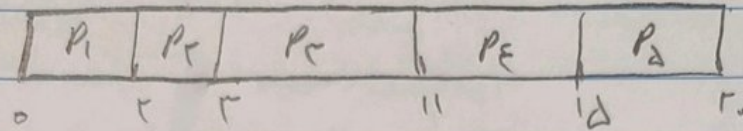
از خود دارند (در کم کنیم به صورت خطی نخواهد بود و مقدار آن کم و کمتر می شود)
 برای مثال با مقدار ۵ سرور مختلف افزایش سرعت را بررسی می کنیم:

$s \backslash Core$	۵%	۲۰%	۵۰%
۲	$\frac{1}{0.05 + \frac{0.95}{2}} = 1.9$	$\frac{1}{\frac{2}{10} + \frac{8}{10}} = 1.47$	$\frac{1}{\frac{1}{4} + \frac{1}{4}} = 1.55$
۴	$\frac{1}{0.05 + \frac{0.95}{4}} = 3.48$	$\frac{1}{\frac{2}{10} + \frac{2}{10}} = 2.5$	$\frac{1}{\frac{1}{4} + \frac{1}{8}} = 1.6$
۸	$\frac{1}{0.05 + \frac{0.95}{8}} = 4.92$	$\frac{1}{\frac{2}{10} + \frac{1}{10}} = 3.33$	$\frac{1}{\frac{1}{4} + \frac{1}{16}} = 1.78$
۱۰۰	$\frac{1}{0.05 + \frac{0.95}{100}} = 19.8$	$\frac{1}{\frac{2}{10} + \frac{1}{10 \times 100}} = 4.8$	$\frac{1}{\frac{1}{4} + \frac{1}{200}} = 1.98$

یعنی افزایش تعداد سرورهای موجود، افزایش سرعت می بخشد، اما اگرچه در حدی است که با افزایش تعداد سرورهای موجود، مقدار کمتری می شود.

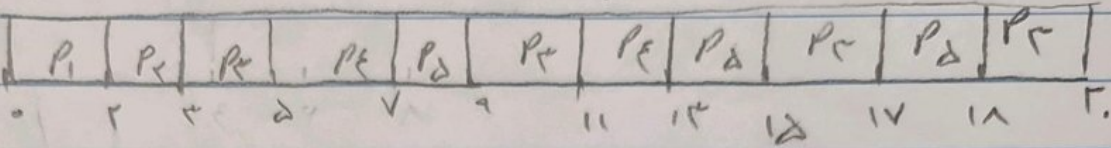
FCFS:

(الف) (د)

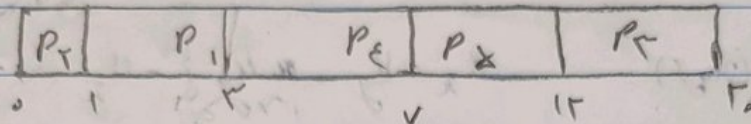


RR:

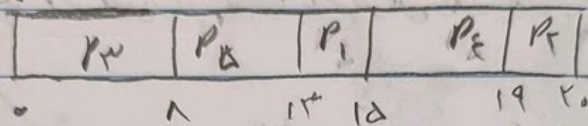
q=2



SFF:



non preemptive:



turn around time:

(ب)

	FCFS	RR	SFF	non Preemptive
P1	2	2	3	18
P2	3	3	1	20
P3	11	20	20	8
P4	18	13	5	19
P5	20	18	12	13

waiting time: turn around time - execution time

(ج)

	FCFS	RR	SFF	non Preemptive
P1	0	0	1	13
P2	2	2	0	19
P3	3	12	12	0
P4	11	9	3	18
P5	18	13	5	8
Sum	34	34	23	58

(ب) در قسمت اول مقدار sum برای هر الگوریتم برابر با مجموع زمان انتظار بود که در اینجا الگوریتم SFF کمترین مقدار را دارد.

(۶) الف) به منظور در اینجا از $P = \text{wait}$ و $V = \text{signal}$ است

با توجه به که تولید کننده، اگر جایی خالی یا برای ابعاد وجود داشته باشد، (یعنی $\text{P(empty)} > 0$) در صفی صفت block می شود و این غیر ضروری است.

ب) با توجه به wait در صفی تولید کننده برای سیگنال block می توان همچنان پیدا کرد تا زمانی که مصرف کننده از صفی پذیرفته نشود، اقدام در جهت برداشتن بسته ها صورت می گیرد.

پ) خرج می توان از این موضوع همچنان حاصل کرد. برای مثال اگر تولید کننده در اجرای خود دوباره با شروع صفی ok در صفی تولید کننده، تولید کننده می تواند صفی ok را به ازای تمام از مصرف کننده بگیرد. اجرا کند یا حتی اگر تولید کننده وارد insert در صفی تولید کننده وارد remove شود، این اتفاق خواهد افتاد، برای جلوگیری می توانیم یک مافز بایستی برای mutex تعریف کنیم.

ت) اگر مافز برای full و empty به درستی انجام نگیرد به بن بست deadlock برس خواهیم.

برای مثال اگر دو تولید کننده داشته باشیم که هر کدام به اندازه n در صفی تولید کننده می توانند جایی خالی بزرگ یا کوچک باشند.

ماده n است اما می توانستیم بود، همچنین در این حالت $\text{block} = 0$ است و صفی تولید کننده می تواند وارد عمل شود.