به نام خدا

دانشگاه صنعتی اصفهان

دانشکده مهندسی برق و کامپیوتر

درس سیستم های عامل

دکتر زینب زالی

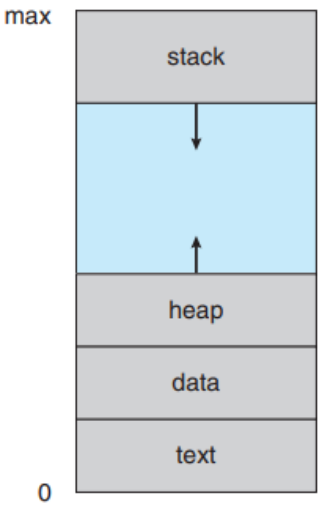
تکلیف دوم

پاسخنامه

سوالاتی که با رنگ قرمز مشخص شده اند، سوالات اختیاری هستند(این سوالات صرفا برای تمرین بیشتر مطرح شده‌اند و امتیازی نیستند)

سوال ۱

الف) چیدمان حافظه مربوط به یک پروسس را با رسم شکل نشان داده و به صورت مختصر توضیح دهید هر بخش به ذخیره چه مقادیری اختصاص دارد؟



Text: شامل دستورات اجرایی برنامه است.

Data: برای ذخیره متغیرهای سراسری و ثابت‌ها استفاده می‌شود.

Heap: حافظه ای که به صورت پویا در طول زمان اجرای برنامه اختصاص داده می شود.

Stack: ذخیره سازی موقت داده ها هنگام فراخوانی توابع (مانند پارامترهای تابع، آدرس های بازگشتی و متغیرهای محلی)

ب) **context switch** چه زمانی اتفاقی می افتد و نقش **PCB** در آن چیست؟

Context switch به معنی تغییر وظیفه پردازشگر (**CPU core**) از یک پروسس به پروسس دیگر است. برای توقف اجرای پروسس قبلی و اجرای پروسس جدید، سیستم باید وضعیت فعلی را در **PCB** ذخیره کند تا در زمان دیگر دوباره بتواند آن پروسس را اجرا کند و باید **PCB** پروسس جدید را لود کند تا بتواند آن را در **CPU** از جایی که قبلا متوقف شده اجرا کند.

وضعیت پروسس (context) شامل مقادیری مانند ثبات‌های CPU، حالت پروسس، و اطلاعات مدیریت حافظه است که در **PCB** نگهداری می‌شود.

پ) فرآیند swapping را توضیح دهید. در چه حالتی سیستم این عمل را انجام می دهد؟

Swapping نوعی زمان‌بندی میانجی در برخی سیستم‌های عامل است. ایده اصلی آن این است که گاهی اوقات مفید است یک پروسس را از حافظه خارج کنیم تا تعداد پروسس‌های فعال برای رقابت بر سر CPU کاهش یابد و در نتیجه درجه multiprogramming کمتر شود.

در این روش، پروسس می‌تواند از حافظه به دیسک **(swap out)** شود و وضعیت فعلی آن در دیسک ذخیره شود. سپس، زمانی که نیاز باشد، پروسس می‌تواند دوباره به حافظه بازگردانده شود **(swap in)** و اجرای آن از همان جایی که متوقف شده بود ادامه یابد. Swapping معمولاً زمانی ضروری است که حافظه به طور کامل پر شده باشد و نیاز به آزادسازی آن برای پروسس‌های دیگر باشد.

سوال ۲

الف) پروسس والد به کمک کدام system call می‌تواند اجرای پروسس فرزند را خاتمه دهد؟

پروسس والد می‌تواند با استفاده از سیستم کال kill اجرای پروسس فرزند خود را خاتمه دهد.

ب) ۳ دلیل برای انجام این کار را بیان کنید.

- پروسس فرزند از منابع بیش از حد مجاز استفاده می‌کند
- وظیفه‌ای که به پروسس فرزند محول شده دیگر نیاز نیست
- پروسس والد در حال خاتمه یافتن است و می‌خواهد از پایان یافتن تمام پروسس‌های فرزند مطمئن شوو

پ) پروسس zombie چیست و در چه شرایطی ایجاد می شود؟

پروسس Zombie پروسسی است که اجرایش تمام شده اما هنوز در جدول پروسس‌ها حضور دارد چون پروسس والد هنوز وضعیت خاتمه آن را با wait() دریافت نکرده است. در این حالت منابع پروسس آزاد شده ولی entry مربوط به آن در لیست PCBها باقی مانده است.

ت) پروسس orphan چیست و در چه شرایطی ایجاد می شود؟

پروسس Orphan پروسسی است که والد آن قبل از اتمام اجرای فرزند خاتمه یافته است. اکثر سیستم ها در این حالت پروسس init (با PID=۱) را به عنوان والد جدید این پروسس تعیین می کند تا بتواند وضعیت خاتمه آن را دریافت کند.

ث) برنامه زیر باعث ایجاد پروسس orphan می شود یا zombie؟ چرا؟

این برنامه منجر به ایجاد یک پروسس زامبی می شود. زیرا پروسس فرزند به محض ایجاد شدن، exit را فراخوانی می کند و پروسس فرزند خاتمه می یابد در حالی که پروسس والد منتظر اتمام پروسس فرزند نمی ماند و ۶۰ ثانیه می خوابد. بنابراین وضعیت خروج فرزند هرگز توسط والد خوانده نمی شود و اطلاعات فرزند همچنان در جدول پروسس باقی می ماند درحالی که فرزندی وجود ندارد.

```c
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>

int main()
{

    pid_t child_pid = fork();

    // Parent process
    if (child_pid > 0)
        sleep(60);

    // Child process
    else
        exit(0);

    return 0;
}
```

سوال ۳

الف) تفاوت program و process را توضیح دهید.

Program: فایلی حاوی دستورالعمل‌ها و داده‌های ذخیره شده روی دیسک است که به صورت passive (غیرفعال) است.

Process: نمونه‌ای در حال اجرا از یک program است که به عنوان یک موجودیت فعال (active) در حافظه قرار دارد.

ب) state های ممکن برای یک پروسس را نام ببرید و توضیح دهید.

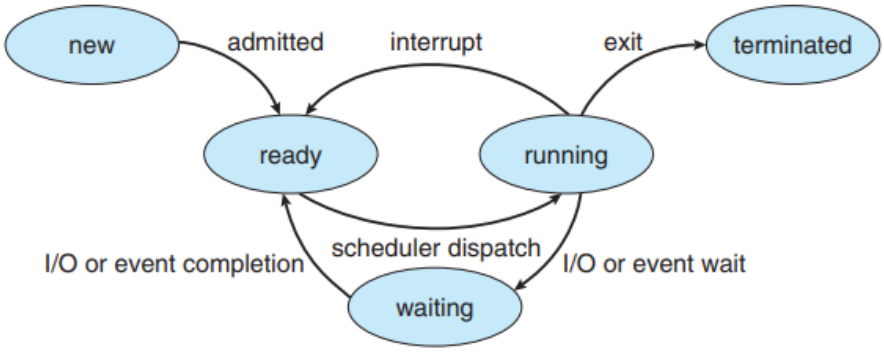
New: پروسس تازه ایجاد شده است

Running: دستورالعمل‌ها در حال اجرا در CPU هستند

Ready: پروسس آماده اجراست و منتظر تخصیص CPU است

Waiting: پروسس منتظر رخداد خاصی است (مثل تکمیل I/O)

Terminated: پروسس اجرایش تمام شده است



پ) عاملی را نام ببرید که باعث می شود پروسس ای که درون cpu در حال اجراست ، از cpu خارج شده و در حالت ready قرار بگیرد.

- پایان یافتن time slice پروسس در سیستم‌های time sharing

ت) برای تغییر وضعیت یک پروسس از حالت اجرا (running) به انتظار (waiting) نام ببرید.

- پروسس یک فرزند ایجاد کرده و با فراخوانی سیستم کال wait به wait queue منتقل می شود تا فرزندش خاتمه یابد.
- پروسس فراخوانی I/O انجام داده و به wait queue تا زمانی که پاسخ I/O را دریافت کند.

در هر دو این حالت ها ، نهایتا پروسس به ready queue منتقل می شود.

سوال ۴

الف) تفاوت بین دو مدل ارتباط بین فرایندی shared memory و message passing را از نظر سربار و سهولت برنامه نویسی مقایسه کنید.

سربار (Overhead)

Shared memory سربار کمتری دارد زیرا فقط در زمان ایجاد ناحیه حافظه مشترک نیاز به system call دارد و پس از آن، دسترسی‌ها مانند دسترسی به حافظه معمولی است. Message passing سربار بیشتر دارد چون برای هر تبادل پیام نیاز به system call و دخالت kernel است,

سهولت برنامه نویسی

Shared memory از لحاظ برنامه نویسی و پیاده سازی پیچیده‌تر است زیرا:

برنامه‌نویس باید از race condition جلوگیری کند

نیاز به مدیریت دقیق دسترسی همزمان به داده‌ها دارد.

ب) چند کاربرد از pipe در ابزارها یا سرویس‌های معروف و متداول پیدا کنید و توضیح دهید در مثال مورد نظر شما از چه نوع پایپی و به چه صورت استفاده می‌شود؟

خط فرمان یونیکس/لینوکس (Unix/Linux Shell)

• توضیح: یکی از معروف‌ترین کاربردهای پایپ در خط فرمان سیستم‌های یونیکس و لینوکس است. از پایپ (|) برای انتقال خروجی یک دستور به ورودی دستور دیگر استفاده می‌شود.

• نوع پایپ: Ordinary Pipe

• مثال:

```
ls -l | grep ".txt"
```

ابزارهای پردازش متن مثل awk, sed و sort

• توضیح: این ابزارها معمولاً در کنار دستورات دیگر استفاده می‌شوند تا داده‌های متنی را پردازش کنند. داده‌ها از طریق پایپ بین این ابزارها منتقل می‌شوند.

• نوع پایپ: Ordinary Pipe

• مثال:

```
cat file.txt | sort | uniq
```

در این مثال، خروجی دستور cat file.txt به ورودی sort منتقل می‌شود و سپس خروجی sort به ورودی uniq داده می‌شود تا تکرارها حذف شوند.

استفاده از پایپ‌ها در سرویس‌های لاگ‌گیری

• توضیح: بسیاری از سیستم‌های لاگ‌گیری و تحلیل داده، از پایپ‌ها برای انتقال داده‌های لاگ به سرویس‌های پردازشی استفاده می‌کنند. این داده‌ها می‌توانند به سرویس‌هایی مثل **syslog** یا ابزارهایی برای پردازش و فیلتر کردن لاگ‌ها منتقل شوند.

• نوع پایپ: معمولاً از **Named Pipe** استفاده می‌شود.

• یک سرویس می‌تواند خروجی لاگ‌های خود را به یک پایپ نام‌گذاری‌شده ارسال کند و ابزار پردازش لاگ، داده‌ها را از این پایپ بخواند و آن‌ها را تحلیل کند.

پ) تفاوت های **Ordinary pipe** و **named pipe** را توضیح دهید.

Ordinary Pipe

موقتی است و فقط تا زمانی که پروسس‌ها در حال ارتباط هستند وجود دارد.

فقط بین پروسس‌های **parent-child** قابل استفاده است

یک‌طرفه (**unidirectional**) است و برای ارتباط دوطرفه نیاز به دو **pipe** است

به عنوان یک نوع خاص از فایل در **UNIX** شناخته می‌شود

با استفاده از تابع **()pipe** ایجاد می‌شود و از طریق **system call** ها **()read** و **()write** قابل دسترسی است

Named Pipe

در فایل سیستم به صورت یک فایل عادی ظاهر می‌شود

تا زمانی که به صورت صریح از فایل سیستم پاک نشود، باقی می‌ماند

بین چندین پروسس (بدون نیاز به رابطه **parent-child**) قابل استفاده است

قابلیت ارتباط دوطرفه (**bidirectional**) را دارد اما فقط به صورت **half-duplex**

برای ارتباط دوطرفه کامل معمولاً از دو **FIFO** استفاده می‌شود

با استفاده از تابع **()mkfifo** ایجاد می‌شود

چندین نویسنده می‌توانند همزمان از آن استفاده کنند.

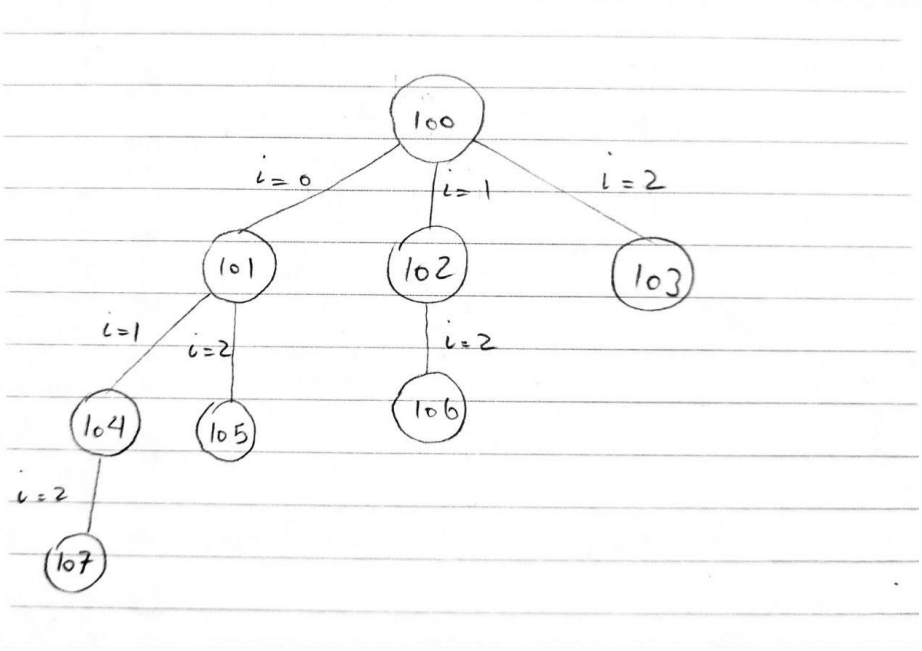
سوال ۵

برنامه زیر را در نظر بگیرید. فرض کنید پروسس اولیه که توسط **shell** ایجاد می شود **pid** برابر با ۱۰۰ دارد و همچنین **pid** های مربوط به پروسس های بعدی به ترتیب با افزایش یک واحدی بدست می‌آیند. فرض کنید

زمانبند سیستم (scheduler) پس از اجرای هر کدام از system call های {fork , wait , return}، از بین پروسس ها آماده برای اجرا ، پروسسی با کمترین pid را انتخاب می کند تا اجرا شود. (cpu تک هسته ای).

```c
int main(){
  int i, pid;
  for(i=0;i<=2;i++){
    fork();
  printf("%d\n",getpid());
  }
  pid = wait();
  printf("%d\n",pid);
  return(0);
}
```

الف) درخت پروسس ها را ترسیم کنید.



ب) آنچه در ترمینال چاپ می‌شود را بنویسید.

100 (i = 0)

100 (i = 1)

100 (i = 2 , 100 wait)

101 (i = 0)

101 (i = 1)

101 (i = 2 , 101 wait)

102 (i = 1)

102 (i = 2 , 102 wait)

103 (i = 2)

-1 (103 exit , 100 ready)

103 (100 exit)

104 (i = 1)

104 (i = 2 , 104 wait)

105 (i = 2)

-1 (105 exit , 101 ready)

105 (101 exit)

106 (i = 2)

-1 (106 exit , 102 ready)

106 (102 exit)

107 (i = 2)

-1 (107 exit , 104 ready)

107 (104 exit)

پ) ترتیب به اتمام رسیدن پروسس ها به چه نحو است؟

به ترتیب :

103

100

105

101

106

102

107

سوال ۶

برنامه زیر را در نظر بگیرید. خروجی این کد چه خواهد بود؟ توضیح دهید.

```c
#include <stdio.h>
#include <unistd.h>

int global_var = 10;

int main() {
    int local_var = 20;

    pid_t pid = fork();

    if (pid == 0) {
        global_var += 10;
        local_var += 10;
        printf("child -> global_var: %d, local_var: %d\n", global_var, local_var);
    } else if (pid > 0) {
        sleep(10);
        printf("parent -> global_var: %d, local_var: %d\n", global_var, local_var);
    }

    return 0;
}
```

خروجی کد:

```
child -> global_var: 20, local_var: 30
parent -> global_var: 10, local_var: 20
```

در اینجا، مقدار global_var و local_var در پروسس والد همچنان به صورت مقدار اولیه‌شان (۱۰ و ۲۰) باقی مانده‌اند، در حالی که در پروسس فرزند این دو مقدار به ۲۰ و ۳۰ افزایش یافته زیرا تغییرات در پروسس فرزند روی پروسس والد تاثیری ندارد و فضای حافظه این دو از هم مجزا است.

سوال ۷

الف)ترد و پروسس را از دیدگاه های متفاوت بررسی کنید(تعریف، استفاده از منابع، امنیت، ...)

تعریف

Process (فرآیند): یک فرآیند، برنامه‌ای است که در حال اجرا است و شامل کد برنامه، داده‌ها و منابعی است که سیستم به آن اختصاص داده است. هر فرآیند می‌تواند شامل یک یا چند **Thread** باشد و هر فرآیند فضای آدرس مستقل خود را دارد.

Thread (رشته): یک **Thread** واحد کوچک‌تری از فرآیند است که جریان کنترلی جداگانه‌ای دارد و می‌تواند به صورت مستقل اجرا شود. همه **Thread** های یک فرآیند، فضای آدرس و منابع آن فرآیند را به اشتراک می‌گذارند.

استفاده از منابع

Process: هر فرآیند فضای آدرس و حافظه خود را دارد و به سیستم‌عامل نیاز دارد تا منابع (مانند حافظه و زمان CPU) را به آن اختصاص دهد. ایجاد یا خاتمه دادن به یک فرآیند هزینه بیشتری نسبت به ایجاد یا خاتمه دادن به یک **Thread** دارد، زیرا تخصیص منابع بیشتری نیاز دارد.
Thread: تمام **Thread** های یک فرآیند، حافظه و منابع فرآیند را به اشتراک می‌گذارند. این اشتراک‌گذاری باعث می‌شود که ایجاد و نابودی یک **Thread** نسبت به فرآیند سریع‌تر و کارآمدتر باشد. همچنین، **Thread** ها به راحتی می‌توانند اطلاعات را بین یکدیگر به اشتراک بگذارند.

امنیت و پایداری

Process: چون فرآیندها فضای آدرس جداگانه دارند، اگر یک فرآیند با مشکلی مواجه شود یا دچار خطا شود، معمولاً تاثیری بر سایر فرآیندها نمی‌گذارد. این جداسازی باعث افزایش امنیت می‌شود، زیرا فرآیندها نمی‌توانند به راحتی به حافظه یا داده‌های فرآیندهای دیگر دسترسی داشته باشند.
Thread: به دلیل اشتراک‌گذاری فضای آدرس بین **Thread**ها، اگر یک **Thread** دچار خطا شود یا دچار مشکل شود، ممکن است کل فرآیند تحت تاثیر قرار گیرد و حتی باعث خرابی کل فرآیند شود. به همین دلیل، مدیریت همزمانی (**Synchronization**) و جلوگیری از شرایط رقابتی (**Race Conditions**) در **Thread**ها اهمیت زیادی دارد.

ارتباط و انتقال داده‌ها

Process : ارتباط بین فرآیندها (**IPC - Inter-Process Communication**) پیچیده‌تر و کندتر است، زیرا فرآیندها فضای آدرس جداگانه دارند و باید از روش‌های خاصی مانند سوکت‌ها، صف‌ها (**Queues**)، یا اشتراک‌گذاری حافظه استفاده کنند.
Thread : ارتباط بین **Thread**ها ساده‌تر و سریع‌تر است، زیرا آن‌ها می‌توانند به طور مستقیم به حافظه مشترک فرآیند دسترسی داشته باشند. با این حال، این اشتراک‌گذاری نیاز به مدیریت دقیق برای جلوگیری از مشکلات مربوط به همزمانی دارد.

کارایی (**Performance**)

Process: به دلیل سربار بیشتری که ایجاد فرآیندها و مدیریت منابع مستقل آن‌ها دارد، اجرای فرآیندها معمولاً زمان بیشتری می‌برد و کارایی کمتری دارد. در مواردی که به جداسازی کامل نیاز باشد، استفاده از فرآیندها مناسب‌تر است.

Thread: چون **Thread** ها منابع را به اشتراک می‌گذارند، زمان کمتری برای ایجاد و مدیریت آن‌ها نیاز است، و عملکرد بهتری در اجرای همزمان دارند. در کاربردهایی که نیاز به عملیات سریع و سبک دارند، استفاده از **Thread** ها مناسب‌تر است.

ب) مزیت ایجاد ترد نسبت به پروسس را بیان کنید و بگویید با وجود این مزیت‌ها چرا از برنامه‌های **multiprocess** استفاده می‌شود؟

۱. سرعت و عملکرد بهتر: ایجاد و مدیریت **Thread** ها سریع‌تر از فرآیندها است و باعث افزایش کارایی می‌شود.

۲. مصرف حافظه کمتر: چون **Thread** ها منابع را به اشتراک می‌گذارند، برنامه‌های مولتی ترد حافظه کمتری مصرف می‌کنند.

۳. ارتباط ساده‌تر: انتقال داده بین **Thread** ها سریع و آسان است، چون از حافظه مشترک استفاده می‌کنند.

۴. واکنش‌پذیری بالاتر: برنامه‌های مولتی ترد می‌توانند سریع‌تر به ورودی‌های کاربر پاسخ دهند.

چرا گاهی مولتی پروسس ترجیح داده می‌شود؟

۱. ایزولاسیون و امنیت: فرآیندها فضای آدرس جداگانه دارند، بنابراین خطای یک فرآیند بر دیگران تاثیر نمی‌گذارد.

۲. پایداری: خرابی یک فرآیند باعث مختل شدن کل برنامه نمی‌شود.

۳. استفاده از فرآیندها می‌تواند به بهره‌وری بهینه از هسته‌های **CPU** کمک کند، چراکه هر فرآیند به‌طور مستقل و جداگانه اجرا می‌شود و سیستم‌عامل می‌تواند آن‌ها را به هسته‌های مختلف اختصاص دهد، که باعث می‌شود استفاده از چند هسته به‌طور قطعی امکان‌پذیر باشد.

در مقابل، در برنامه‌های چندترد (**Multi-Thread**)، اگرچه امکان اجرای تردها به‌طور موازی روی چند هسته وجود دارد، اما این موضوع به نحوه پیاده‌سازی لایبرری ترد بستگی دارد. اگر لایبرری ترد به‌گونه‌ای پیاده‌سازی شده باشد که هر ترد سطح کاربر به یک ترد سطح کرنل اختصاص داده شود، امکان موازی‌سازی روی چند هسته فراهم خواهد شد.

بنابراین

• در برنامه‌های مالتی پروسس (Multi-Process): موازی‌سازی روی چند هسته به‌صورت قطعی امکان‌پذیر است.

• در برنامه‌های مالتی ترد (Multi-Thread): موازی‌سازی روی چند هسته به پیاده‌سازی لایبرری ترد بستگی دارد و ممکن است تضمین‌شده نباشد.

۴. مدیریت بهتر منابع: برنامه‌های بزرگ و پیچیده می‌توانند منابع را به طور جداگانه مدیریت کنند.

سوال ۸

به سوالات زیر پاسخ دهید.

الف) task ها میتوانند به دو صورت موازی(parallelism) و هم روند(concurrency) اجرا شوند.تفاوت این دو روش را با رسم شکل در صورت وجود بیش از یک core توضیح دهید.

همروندی (Concurrency)

در همروندی، چندین تسک می‌توانند به طور همزمان پیشرفت کنند، اما لزوماً به صورت واقعی و همزمان اجرا نمی‌شوند. سیستم با استفاده از یک برنامه‌ریز (Scheduler) بین تسک‌ها جابه‌جا می‌شود، و این جابه‌جایی به قدری سریع است که به نظر می‌رسد تسک‌ها به صورت موازی در حال اجرا هستند. با این حال، در هر لحظه، یک هسته فقط می‌تواند یک تسک را اجرا کند. این توهم موازی‌سازی باعث می‌شود کاربر تصور کند که تمام تسک‌ها به صورت همزمان اجرا می‌شوند، اما در واقعیت، اجرای تسک‌ها به صورت متناوب است و هر تسک برای مدت کوتاهی اجرا می‌شود.

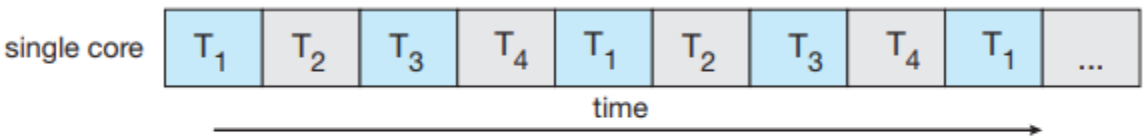


Figure 4.3 Concurrent execution on a single-core system.

موازی‌سازی (Parallelism)

موازی‌سازی، در مقابل، زمانی اتفاق می‌افتد که سیستم بتواند چندین تسک را به صورت واقعی و همزمان روی هسته‌های مختلف اجرا کند. اگر سیستم دارای چندین هسته پردازشی باشد، می‌توان یک تسک را روی یک هسته و تسک دیگر را روی هسته دیگری اجرا کرد، به طوری که این دو تسک به صورت واقعی و بدون وقفه به طور

همزمان اجرا شوند. این نوع اجرا از قدرت چند هسته‌ای بهره می‌گیرد و توانایی پردازش چندین تسک را به صورت موازی فراهم می‌کند

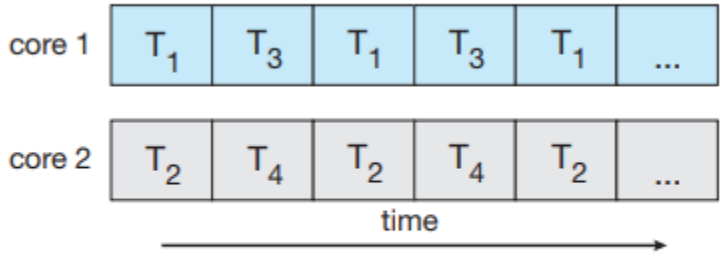


Figure 4.4 Parallel execution on a multicore system.

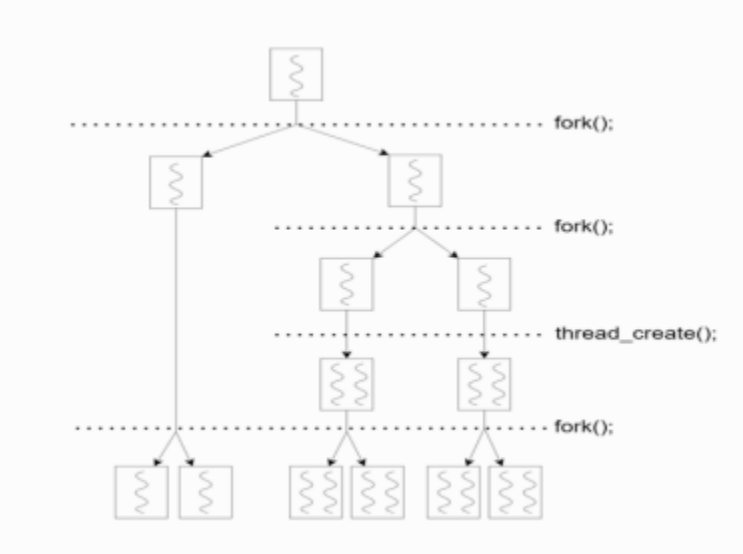
ب) کد زیر را در نظر بگیرید. تابع pthread_create() یک ریسمان جدید را در فرآیند فراخوانی شروع می‌کند. چند فرآیند منحصر به فرد ایجاد می شود؟ چه تعداد رشته منحصر به فرد ایجاد می شود؟

```
pid_t pid = fork();
if(pid == 0){
        fork();
        pthread_create(...);
}
fork();
```

این قطعه کد ۶ فرآیند که شامل خود فرآیند اصلی نیز می‌باشد به همراه ۱۰ رشته تولید می‌کند. (طبق فرض اینکه فراخوانی fork رشته‌های فرآیند صدا زننده را نیز کپی می‌کند. ولی بدون این فرض همین تعداد فرآیند و ۸ رشته را خواهیم داشت.)

- صدا زدن fork اول یک فرآیند دیگر ایجاد می‌کند.
- fork دوم که داخل if قرار دارد تنها توسط فرآیند فرزند که از fork اول ساخته شده است صدا زده می‌شود.
- حال دو فرآیند pthread_create را داخل if صدا می‌زنند و در این لحظه ما ۳ فرآیند و ۵ رشته خواهیم داشت.

- هر ۳ فرآیند **fork** آخر را صدا می‌زنند و این یعنی ۳ فرآیند ما به ۶ فرآیند تبدیل خواهد شد و ۵ رشته به ۱۰ رشته تبدیل خواهد شد.



سوال ۹

کد یک برنامه ساده shell بنویسید که توانایی اجرای برنامه های مختلف به صورت مستقل را داشته باشد و ضمنا عملگرهای زیر را هم داشته باشد.

الف) پایپ دو دستور. برای مثال : ”ls -l | grep “os

ب) ارسال خروجی یک دستور (که در حالت اصلی در خروجی استاندارد چاپ می‌شود) به یک فایل مشخص. برای مثال: ls -l > file.txt

روش پیاده سازی موارد الف و ب را توضیح دهید.

فایل پاسخ در کنار این فایل قرار داده شده.

به نقش و نحوه اثر & در آخر هر **command** هم دقت کنید.

در این سوال نحوه پیاده سازی | با **ordinary pipe** مهمه و همینطور دقت به نحوه بک گراند کردن یک کامند