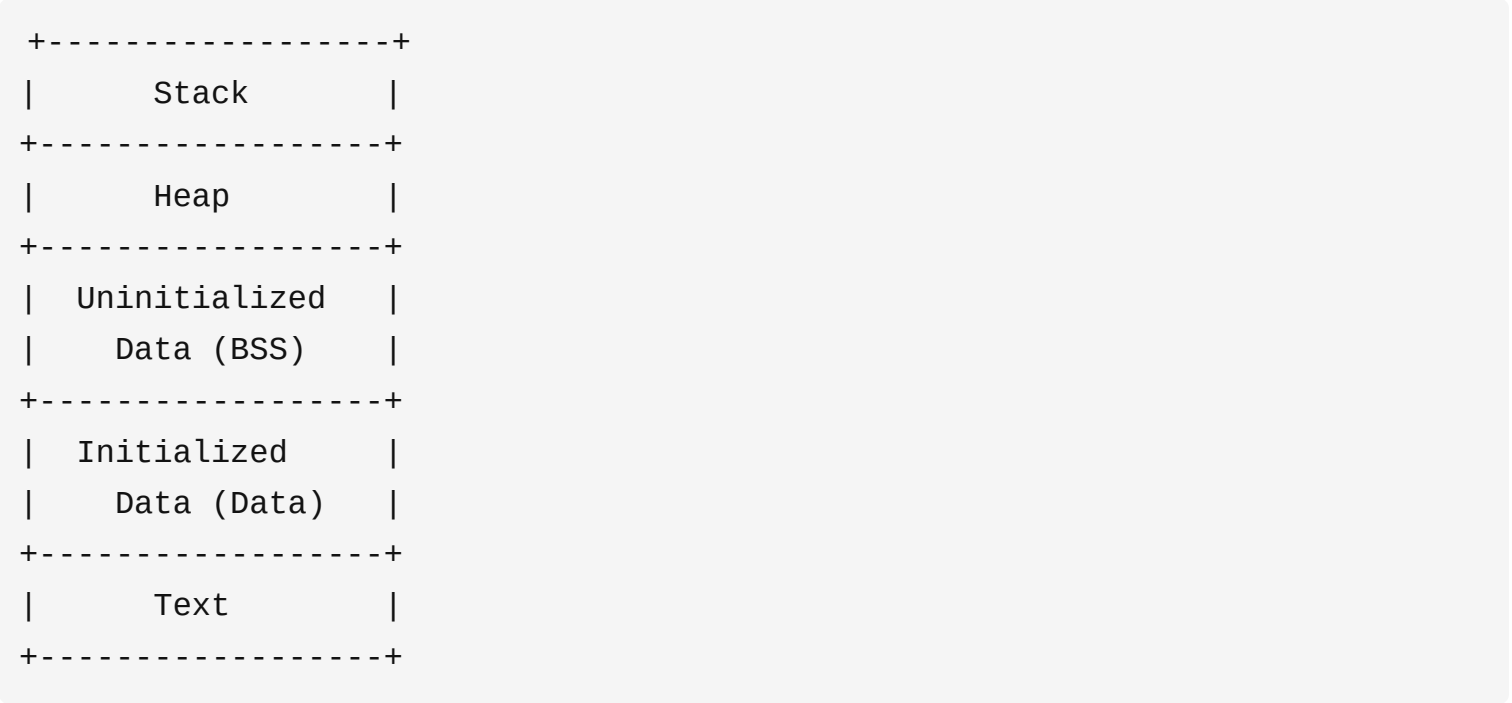
تکلیف 2

سوال 1

الف) چیدمان حافظه مربوط به یک پروسس

چیدمان حافظه یک پروسس به صورت زیر است:



- **Text**: شامل کدهای اجرایی برنامه است.
- **Initialized Data (Data)**: شامل متغیرهای سراسری و استاتیک که مقداردهی اولیه شده‌اند.
- **Uninitialized Data (BSS)**: شامل متغیرهای سراسری و استاتیک که مقداردهی اولیه نشده‌اند.
- **Heap**: برای تخصیص حافظه پویا در زمان اجرا استفاده می‌شود.
- **Stack**: برای ذخیره‌سازی متغیرهای محلی و اطلاعات مربوط به فراخوانی توابع استفاده می‌شود.

ب) Context Switch

Context Switch زمانی اتفاق می‌افتد که سیستم عامل تصمیم می‌گیرد که اجرای یک پروسس را متوقف کرده و پروسس دیگری را اجرا کند. نقش PCB (Process Control Block) در این فرآیند بسیار مهم است. PCB شامل اطلاعاتی مانند شمارنده برنامه، رجیسترها، و وضعیت پروسس است که برای ذخیره و بازیابی وضعیت پروسس‌ها در هنگام Context Switch استفاده می‌شود.

پ) فرآیند Swapping

Swapping فرآیندی است که در آن یک پروسس به طور موقت از حافظه اصلی به حافظه جانبی (مانند دیسک) منتقل می‌شود تا فضای بیشتری برای پروسس‌های دیگر فراهم شود. این عمل زمانی انجام می‌شود که حافظه اصلی پر شده باشد و سیستم نیاز به فضای بیشتری برای اجرای پروسس‌های جدید یا فعال دارد.

سوال 2

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

پروسس والد می‌تواند با استفاده از `kill` system call اجرای پروسس فرزند را خاتمه دهد.

ب) 3 دلیل برای انجام این کار را بیان کنید

1. پروسس فرزند به درستی کار نمی‌کند و باعث ایجاد مشکلات در سیستم می‌شود.
2. پروسس والد نیاز به منابع بیشتری دارد و می‌خواهد منابعی که توسط پروسس فرزند استفاده می‌شود را آزاد کند.
3. پروسس فرزند به پایان رسیده است و دیگر نیازی به اجرای آن نیست.

پ) پروسس zombie چیست و در چه شرایطی ایجاد می‌شود؟

پروسس zombie پروسسی است که اجرای آن به پایان رسیده است ولی هنوز اطلاعات آن در جدول پروسس‌ها باقی مانده است. این حالت زمانی ایجاد می‌شود که پروسس والد هنوز وضعیت خروج پروسس فرزند را نخوانده باشد.

ت) پروسس orphan چیست و در چه شرایطی ایجاد می‌شود؟

پروسس orphan پروسسی است که والد آن قبل از اتمام اجرای پروسس فرزند خاتمه یافته است. در این شرایط، پروسس فرزند به پروسس init (PID 1) تعلق می‌گیرد.

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

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

int main()
{
 pid_t child_pid = fork();

 if (child_pid > 0)
 {
  // Parent process
  sleep(60);
 }
 else if (child_pid == 0)
 {
  // Child process
  exit(0);
 }

 return 0;
}
```

این برنامه باعث ایجاد پروسس zombie می‌شود. زیرا پروسس فرزند به سرعت خاتمه می‌یابد ولی پروسس والد به مدت 60 ثانیه به خواب می‌رود و وضعیت خروج پروسس فرزند را نمی‌خواند.

سوال 3

الف) تفاوت program و process را توضیح دهید

- **Program:** یک برنامه (Program) مجموعه‌ای از دستورات و کدها است که بر روی دیسک ذخیره شده و هنوز در حال اجرا نیست. برنامه‌ها فایل‌های اجرایی هستند که می‌توانند توسط سیستم عامل بارگذاری و اجرا شوند.
- **Process:** یک پروسس (Process) نمونه‌ای از یک برنامه در حال اجرا است. پروسس شامل کد برنامه، داده‌ها، و منابع مورد نیاز برای اجرا می‌باشد. هر پروسس فضای حافظه و منابع مستقل خود را دارد.

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

1. **New:** پروسس تازه ایجاد شده و هنوز آماده اجرا نیست.
2. **Ready:** پروسس آماده اجرا است و منتظر تخصیص CPU می‌باشد.
3. **Running:** پروسس در حال اجرا بر روی CPU است.
4. **Waiting:** پروسس منتظر یک رویداد یا منبع خاص است و نمی‌تواند اجرا شود.
5. **Terminated:** پروسس اجرای خود را به پایان رسانده و منابع آن آزاد شده‌اند.

پ) عاملی را نام ببرید که باعث می‌شود پروسس ای که درون CPU در حال اجراست، از CPU خارج شده و در حالت ready قرار بگیرد

- **Interrupt:** یک وقفه (Interrupt) می‌تواند باعث شود که پروسس در حال اجرا از CPU خارج شده و در حالت ready قرار بگیرد. این وقفه می‌تواند توسط سخت‌افزار یا سیستم عامل ایجاد شود.

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

1. **I/O Request:** درخواست ورودی/خروجی (I/O) می‌تواند باعث شود که پروسس از حالت اجرا به حالت انتظار تغییر وضعیت دهد تا عملیات I/O تکمیل شود.
2. **Resource Request:** درخواست منابعی که در دسترس نیستند، مانند قفل‌ها یا حافظه، می‌تواند باعث شود که پروسس به حالت انتظار برود تا منابع مورد نیاز آزاد شوند.

سوال 4

الف) تفاوت بین message passing و shared memory را در ارتباط بین فرآیندها توضیح دهید

در ارتباط بین فرآیندها (IPC)، دو روش اصلی وجود دارد: message passing و shared memory.

- **Message Passing:** در این روش، فرآیندها با ارسال و دریافت پیام‌ها با یکدیگر ارتباط برقرار می‌کنند. این روش نیاز به هماهنگی سیستم عامل دارد و معمولاً برای ارتباط بین فرآیندهایی که در سیستم‌های توزیع شده اجرا می‌شوند، مناسب است.
- **Shared Memory:** در این روش، یک بخش از حافظه به صورت مشترک بین فرآیندها استفاده می‌شود. فرآیندها می‌توانند به طور مستقیم به این حافظه دسترسی داشته باشند و داده‌ها را به اشتراک بگذارند. این روش نیاز به هماهنگی و همگام‌سازی بین فرآیندها دارد تا از تداخل داده‌ها جلوگیری شود.

ب) نحوه استفاده از pipe در ارتباط بین فرآیندها را توضیح دهید و یک مثال ساده از آن بنویسید

Pipe یکی از روش‌های ارتباط بین فرآیندها است که به فرآیندها اجازه می‌دهد تا داده‌ها را به صورت یک طرفه از یک فرآیند به فرآیند دیگر ارسال کنند. در سیستم‌های یونیکس و لینوکس، pipe با استفاده از تابع `()pipe` ایجاد می‌شود.

مثال ساده از استفاده از pipe:

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

int main() {
  int fd[2];
  pid_t pid;
  char write_msg[] = "Hello, world!";
  char read_msg[20];

  // Create a pipe
  if (pipe(fd) == -1) {
    perror("pipe");
    return 1;
  }

  pid = fork();

  if (pid < 0) {
    perror("fork");
    return 1;
  }

  if (pid > 0) {
    // Parent process
    close(fd[0]); // Close unused read end
    write(fd[1], write_msg, strlen(write_msg) + 1);
    close(fd[1]); // Close write end after writing
  } else {
    // Child process
    close(fd[1]); // Close unused write end
    read(fd[0], read_msg, sizeof(read_msg));
    printf("Child received: %s\n", read_msg);
    close(fd[0]); // Close read end after reading
  }

  return 0;
```

```
}
```

در این مثال، فرآیند والد پیامی را از طریق pipe به فرآیند فرزند ارسال می‌کند و فرآیند فرزند آن را دریافت و چاپ می‌کند.

سوال 5

```
int main(int argc, char const* argv[])
{
 int i, pid;
 for (i = 0; i <= 2; i++)
 {
  fork();
  printf("%d\n", getpid());
  pid = wait(NULL);
  printf("%d\n", pid);
 }

 return 0;
}
```

الف) درخت پروسس‌ها

```
        100
      / | \
    101 102 103
   /|   /|   /|
 104 105 106 107
```

ب) آنچه در ترمینال چاپ می‌شود

```
100
101
-1
101
102
-1
102
103
-1
103
102
101
```

```
104
-1
104
101
100
105
-1
105
106
-1
106
105
100
107
-1
107
```

پ) ترتیب به اتمام رسیدن پروسس‌ها

پروسس 100 شروع می‌شود. پروسس 101 شروع می‌شود. پروسس 101 به پایان می‌رسد (−1). پروسس 102 شروع می‌شود. پروسس 102 به پایان می‌رسد (−1). پروسس 103 شروع می‌شود. پروسس 103 به پایان می‌رسد (−1). پروسس 102 دوباره شروع می‌شود. پروسس 101 دوباره شروع می‌شود. پروسس 104 شروع می‌شود. پروسس 104 به پایان می‌رسد (−1). پروسس 101 دوباره شروع می‌شود. پروسس 100 دوباره شروع می‌شود. پروسس 105 شروع می‌شود. پروسس 105 به پایان می‌رسد (−1). پروسس 106 شروع می‌شود.

```
101, 102, 103, 104, 105, 106, 107
```

سوال 6

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

int gloabal_var = 10;

int main(int argc, char const* argv[])
{
 int local_var = 20;

 pid_t pid = fork();

 if (pid == 0)
 {
```

```
            // Child process
            gloabal_var += 10;
            local_var += 10;
            printf("Child: global_var = %d, local_var = %d\n", gloabal_var,
local_var);
          }
          else if (pid > 0)
          {
            // Parent process
            sleep(10);
            printf("Parent: global_var = %d, local_var = %d\n", gloabal_var,
local_var);
          }

          return 0;
        }
```

توضیح

پروسس والد

- متغیر سراسری `gloabal_var` مقدار اولیه 10 را دارد.
- متغیر محلی `local_var` مقدار اولیه 20 را دارد.
- پس از اجرای `fork()`، پروسس والد به شرط `else if (pid > 0)` می‌رود و یک ثانیه صبر می‌کند تا پروسس فرزند ابتدا اجرا شود.
- سپس مقدار متغیرهای `gloabal_var` و `local_var` را چاپ می‌کند.

پروسس فرزند

- پس از اجرای `fork()`، پروسس فرزند به شرط `if (pid == 0)` می‌رود.
- متغیر سراسری `gloabal_var` را 10 واحد افزایش می‌دهد (مقدار جدید 20).
- متغیر محلی `local_var` را 10 واحد افزایش می‌دهد (مقدار جدید 30).
- سپس مقدار متغیرهای `gloabal_var` و `local_var` را چاپ می‌کند.

خروجی

با توجه به توضیحات بالا، خروجی برنامه به این صورت خواهد بود:

```
Child Process: gloabal_var = 20, local_var = 30
Parent Process: gloabal_var = 10, local_var = 20
```

توضیح خروجی

- پروسس فرزند متغیرهای خود را تغییر می‌دهد و مقادیر جدید را چاپ می‌کند.
- پروسس والد متغیرهای خود را تغییر نمی‌دهد و مقادیر اولیه را چاپ می‌کند.
- هر پروسس کپی جداگانه‌ای از متغیرها دارد و تغییرات در یک پروسس بر دیگری تأثیری ندارد.

سوال 7

الف) ترد و پروسس را از دیدگاه‌های متفاوت بررسی کنید

- تعریف:
  - پروسس: یک برنامه در حال اجرا است که شامل کد برنامه، داده‌ها، و منابع مورد نیاز برای اجرا می‌باشد.
  - ترد: واحد اجرایی کوچکتری است که در داخل یک پروسس اجرا می‌شود و منابع پروسس را به اشتراک می‌گذارد.
- استفاده از منابع:
  - پروسس: هر پروسس دارای فضای حافظه و منابع مستقل خود است.
  - ترد: تردها حافظه و منابع پروسس والد خود را به اشتراک می‌گذارند.
- امنیت:
  - پروسس: پروسس‌ها از یکدیگر جدا هستند و دسترسی به حافظه یکدیگر ندارند، که امنیت بیشتری فراهم می‌کند.
  - ترد: تردها به حافظه مشترک دسترسی دارند، که می‌تواند منجر به مشکلات امنیتی و تداخل داده‌ها شود.
- زمان‌بندی و مدیریت:
  - پروسس: ایجاد و مدیریت پروسس‌ها هزینه‌برتر است و زمان بیشتری می‌برد.
  - ترد: تردها سبک‌تر هستند و مدیریت آن‌ها سریع‌تر و کم‌هزینه‌تر است.

ب) مزیت ایجاد ترد نسبت به پروسس و دلایل استفاده از برنامه‌های multiprocess

- مزیت‌های ترد:
  - کارایی بالاتر: تردها به دلیل اشتراک منابع و حافظه، سریع‌تر از پروسس‌ها ایجاد و مدیریت می‌شوند.
  - ارتباط سریع‌تر: تردها به دلیل اشتراک حافظه، می‌توانند سریع‌تر با یکدیگر ارتباط برقرار کنند.
  - استفاده بهینه از منابع: تردها منابع کمتری نسبت به پروسس‌ها مصرف می‌کنند.
- دلایل استفاده از برنامه‌های multiprocess:
  - پایداری و امنیت: پروسس‌ها از یکدیگر جدا هستند و خرابی یک پروسس تأثیری بر پروسس‌های دیگر ندارد.
  - استفاده از چند هسته‌ای: برنامه‌های multiprocess می‌توانند از چند هسته پردازنده به طور همزمان استفاده کنند و کارایی را افزایش دهند.
  - مدیریت بهتر منابع: در برخی موارد، استفاده از پروسس‌ها برای مدیریت منابع و جلوگیری از تداخل داده‌ها مناسب‌تر است.

سوال 8

الف) تفاوت بین Parallelism و Concurrency

در سیستم‌های چند هسته‌ای، taskها می‌توانند به دو صورت موازی (Parallelism) و هم‌روند (Concurrency) اجرا شوند. تفاوت این دو روش به شرح زیر است:

- **Parallelism**: در این روش، taskها به طور همزمان و واقعی بر روی چندین هسته پردازنده اجرا می‌شوند. هر هسته یک task را اجرا می‌کند و این باعث افزایش کارایی و سرعت اجرای برنامه می‌شود.
- **Concurrency**: در این روش، taskها به صورت هم‌روند اجرا می‌شوند، اما لزوماً به طور همزمان بر روی چندین هسته پردازنده اجرا نمی‌شوند. در واقع، taskها به نوبت و با استفاده از زمان‌بندی سیستم عامل بر روی یک یا چند هسته اجرا می‌شوند.

شکل تفاوت بین Parallelism و Concurrency

```
Parallelism (موازی):
+---------+---------+
| Task 1  | Task 2  |  Core 1
+---------+---------+
| Task 3  | Task 4  |  Core 2
+---------+---------+

Concurrency (هم‌روند):
+---------+---------+
| Task 1  | Task 2  |  Core 1
+---------+---------+
| Task 3  | Task 4  |  Core 1
+---------+---------+
```

در Parallelism، taskها به طور همزمان بر روی هسته‌های مختلف اجرا می‌شوند، در حالی که در Concurrency، taskها به نوبت بر روی یک یا چند هسته اجرا می‌شوند.

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 می‌شود؟ چه تعداد رشته منحصر به فرد ایجاد می‌شود؟

```
pid_t pid = fork();
if (pid == 0)
{
 fork();
 pthread_create(&thread, NULL, thread_function, NULL);
}
fork();
```

در این کد، تعداد فرآیندها و رشته‌های منحصر به فرد به شرح زیر است:

- فرآیندها:
  - اولین `fork()` دو فرآیند ایجاد می‌کند.
  - دومین `fork()` در بلوک `if (pid == 0)` یک فرآیند دیگر ایجاد می‌کند.
  - سومین `fork()` سه فرآیند دیگر ایجاد می‌کند.

- در مجموع، 6 فرآیند منحصر به فرد ایجاد می‌شود.

- رشته‌ها:
  - تابع `pthread_create()` یک رشته جدید در فرآیند فرزند دومین `fork()` ایجاد می‌کند و یک رشته دیگر در نوه فرآیند اول ایجاد می‌شود.
  - بنابراین، 2 رشته منحصر به فرد ایجاد می‌شود.

با توجه به خروجی داده شده:

- فرآیندهای منحصر به فرد: 6
- رشته‌های منحصر به فرد: 2

سوال 9

کد برنامه ساده Shell

```
#!/bin/bash

# Function to execute a command
execute_command() {
  eval "$1"
}

# Function to execute a piped command
execute_piped_command() {
  eval "$1 | $2"
}

# Function to redirect output of a command to a file
redirect_output_to_file() {
  eval "$1 > $2"
}

# Main script logic
echo "Choose an option:"
echo "1. Execute a command"
echo "2. Execute a piped command"
echo "3. Redirect output of a command to a file"
read -p "Enter your choice: " choice

case $choice in
1)
  read -p "Enter the command to execute: " cmd
  execute_command "$cmd"
```

```
        ;;
    2)
        read -p "Enter the first command: " cmd1
        read -p "Enter the second command: " cmd2
        execute_piped_command "$cmd1" "$cmd2"
        ;;
    3)
        read -p "Enter the command: " cmd
        read -p "Enter the file name: " file
        redirect_output_to_file "$cmd" "$file"
        ;;
    *)
        echo "Invalid choice"
        ;;
    esac
```

روش پیاده‌سازی

- الف) پایپ دو دستور:
  - در این روش، خروجی دستور اول به عنوان ورودی دستور دوم استفاده می‌شود. برای مثال، `"ls -l | grep "os` لیست فایل‌ها را فیلتر می‌کند تا فقط فایل‌هایی که شامل "os" هستند نمایش داده شوند.
  - در کد بالا، تابع `execute_piped_command` برای اجرای دستورات پایپ شده استفاده می‌شود. این تابع دو دستور را به عنوان ورودی می‌گیرد و با استفاده از `eval` و عملگر `|` آن‌ها را اجرا می‌کند.
- ب) ارسال خروجی یک دستور به یک فایل:
  - در این روش، خروجی یک دستور به جای نمایش در خروجی استاندارد، به یک فایل مشخص ارسال می‌شود. برای مثال، `ls -l > file.txt` خروجی دستور `ls -l` را در فایل `file.txt` ذخیره می‌کند.
  - در کد بالا، تابع `redirect_output_to_file` برای انجام این کار استفاده می‌شود. این تابع دستور و نام فایل را به عنوان ورودی می‌گیرد و با استفاده از `eval` و عملگر `>` خروجی دستور را به فایل مشخص شده هدایت می‌کند.