پاسخ تکلیف دوم سیستم عامل

سوال ۱

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

پاسخ: به کمک سیستم کال ()kill.

ب) ۳ دلیل برای انجام این کار را بیان کرده و هریک را توضیح دهید.

پاسخ:

۱. در برخی موارد، فرآیند فرزند ممکن است بیشتر از منابعی (مانند حافظه ویا CPU) که به آن در ابتدا اختصاص داده شده ، استفاده کند. این اتفاق می تواند باعث کاهش عملکرد سیستم شود. بنابراین فرایند والد به کمک دستور ()kill میتواند از این اتفاق جلوگیری کند.

۲. ممکن است تسکی که به فرزند اختصاص داده شده، دیگر موردنیاز نباشد.
فرآیند والد ممکن است در ابتدا فرآیندهای فرزند را برای انجام وظایف خاص ایجاد کرده باشد، اما به دلیل تغییر شرایط یا نیازمندی ها، آن وظایف دیگر ضروری نیست.
به طور مثال، یک برنامه پردازش فایل را در نظر بگیرید که در آن والد یک فرایند فرزند را برای فشرده سازی یک فایل ایجاد می کند. اگر به دلایلی والد تصمیم بگیرد که آن فایل دیگر مورد نیاز نیست، ممکن است فرآیند فرزند را خاتمه دهد.

۳. اگر فرایند والد در حالت خروج باشد و بخواهد وارد terminated state شود، سیستم عامل به فرزند اجازه ی ادامه ی فعالیت نمیدهد.
برخی از سیستم‌عامل‌ها دارای policyای هستند که تمام فرآیندهای فرزند را هنگامی که فرآیند والد آنها خارج می‌شود، خاتمه می‌دهد. این کار تضمین می‌کند که هیچ فرآیند یتیمی باقی نمی‌ماند.

سوال ۲

الف) پروسس zombie چیست و در چه شرایطی ایجاد میشود.

پاسخ:
هنگامی که یک فرآیند اجرای خود را کامل کرد یا خاتمه یافت، یک سیگنال (SIGCHLD در سیستم های لینوکس) به فرآیند والد خود ارسال می کند. سیگنال به والد اطلاع می‌دهد که فرآیند فرزند اجرای خود را به پایان رسانده است.
پس از آن، فرایند فرزند بلافاصله وارد حالت زامبی می شود تا زمانی که فرآیند والد به آن رسیدگی کند و به اصطلاح فرایند فرزند را درو کند.
(Reaping به معنای خواندن اطلاعاتی مانند وضعیت خروج، شناسه فرآیند و غیره فرآیند فرزند توسط والد است.)
به طور خلاصه، می توان گفت که یک فرآیند زامبی یک فرآیند خاتمه یافته یا تکمیل شده است که در جدول فرآیند (process table)باقی می ماند و تا زمانی که فرآیند والد به وضعیت آن رسیدگی کند، وجود خواهد داشت.
فرایند والد این کار را با فراخوانی سیستم کال ()wait برای فرایند فرزند و خواندن مقدار خروجی آن انجام می دهد.

ب) پروسس orphan چیست و در چه شرایطی ایجاد میشود.

پاسخ:
فرآیند یتیم فرآیندی است که هنوز در حال اجراست اما فرآیند والد آن دیگر وجود ندارد (یا کشته شده یا خارج شده است).
زمانی که یک فرایند یتیم ایجاد میشود، سیستم عامل مسئولیت آن فرایند را به عهده میگیرد.
ورژن های قدیمی تر یونیکس این مساله را با تخصیص فرآیند init به‌عنوان والد جدید به فرآیندهای یتیم حل میکردند.
فرآیند init به صورت دوره‌ای ()waitرا فراخوانی می‌کند، در نتیجه اجازه می‌دهد وضعیت خروج از هر فرآیند یتیم جمع‌آوری شود و شناسه فرآیند یتیم و ورودی مربوط به ان از جدول فرایند پاک شود.
اگرچه اکثر سیستم‌های لینوکس init را با systemd(که pid=1 است) جایگزین کرده‌اند، فرآیند systemd همچنان می‌تواند همان نقش را ایفا کند.
لینوکس همچنین به فرآیندهای غیر از systemd اجازه می‌دهد تا فرآیندهای یتیم را به ارث ببرند و خاتمه آنها را مدیریت کنند.

برنامه ۱

```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;
}
```

برنامه ۲

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

int main()
{
    int pid;
    pid = fork();

    if (pid == 0)
    {
        printf("I am the child, my process ID is %d\n", getpid());
        printf("My parent's process ID is %d\n", getppid());
        sleep(30);
        printf("\nAfter sleep\nI am the child, my process ID is %d\n", getpid());
        printf("My parent's process ID is %d\n", getppid());
        exit(0);
    }
    else
    {
        sleep(20);
        printf("I am the parent, my process ID is %d\n", getpid());
        printf("The parent's parent, process ID is %d\n", getppid());
        printf("Parent terminates\n");
    }
    return 0;
}
```

ج) در ۲برنامه ، مشخص کنید هریک باعث ایجاد کدام یک از پروسس های zombie و یا orphan میشود. علت انتخاب خود را توضیح دهید.

پاسخ:

۱. کد برنامه ۱ مربوط به ساخت یک فرایند زامبی است.
فرایند فرزند با استفاده از سیستم کال exit() اجرای خود را کامل می کند.
بنابراین هنگامی که فرزند اجرای خود را به پایان می رساند سیگنال "SIGCHLD "توسط کرنل به فرآیند والد ارسال می شود.
در حالت ایده آل، والد باید وضعیت فرزند را از جدول فرآیند بخواند و سپس سطر مربوط به فرزند را حذف کند.
اما در اینجا والد منتظر نمی ماند تا فرزند خاتمه یابد، بلکه کار بعدی خود را انجام می دهد، یعنی 60 ثانیه می خوابد.
بنابراین وضعیت خروج فرزند هرگز توسط والد خوانده نمی شود و اطلاعات فرزند همچنان در جدول فرآیند باقی می ماند حتی زمانی که فرزند دیگر وجود ندارد .

۲. کد برنامه ۲ یک فرایند یتیم را نشان میدهد.
در این کد، فرآیند والد 20 ثانیه می خوابد در حالی که فرایند فرزند 30 ثانیه می خوابد.
بنابراین پس از 20 ثانیه خواب، اجرای والد کامل می شود در حالی که فرآیند فرزند حداقل برای 30 ثانیه هنوز وجود دارد.
هنگامی که فرآیند فرزند به یک فرآیند یتیم تبدیل می‌شود، کرنل یک فرآیند را به عنوان والد به فرآیند فرزند اختصاص می‌دهد.
در نتیجه ایدی والد ، قبل و بعد از sleep() متفاوت خواهد بود.
خروجی اجرای برنامه:

```
hadis@ubuntu:~/Documents/Os_TA/HW2$ ./orphan
I am the child, my process ID is 5601
My parent's process ID is 5600
I am the parent, my process ID is 5600
The parent's parent, process ID is 5550
Parent terminates
hadis@ubuntu:~/Documents/Os_TA/HW2$
After sleep
I am the child, my process ID is 5601
My parent's process ID is 2416
```

سوال ۳

الف) تفاوت بین دو مدل ارتباطی Shared memory و message passing برای ارتباطات بین فرآیندی (IPC) را بیان کنید.

پاسخ:

۱. هر دو این روشها میتوانند برای ارتباط بین دو فرآیند روی یک سیستم مشترک استفاده شود. اما روش دوم گاهی برای ارتباط بین دو فرآیند روی دو سیستم مختلف هم قابل استفاده است.

۲. shared memory یک بافر اشتراکی بین دو فرآیند در نظر میگیرد که هر دو به این بافر دسترسی دارند و از طریق خواندن این بافر یا تغییر مقادیر این بافر با هم در ارتباط هستند. در صورتی که در روشmessage passing بین دو فرآیند کانال ارتباطی ایجاد میشود که هر یک از پروسسها میتواند از طریق این کانال برای پروسس دیگر محتوایی را در قالب پیام ارسال کند.

۳. پس در روش اول مدیریت بافر اشتراکی (آیا اطلاعات جدیدی در بافر قرار گرفته یا نه؟ آیا اطلاعاتی که در بافر است قبلا خوانده شده یا نه؟ آیا بافر جا دارد که اطلاعات جدیدی در آن قرار داده شود؟ مشکلات همزمانی دسترسی به بافر و ....) توسط خود دو فرآیند باید کنترل شود در صورتی که در روش دوم کتابخانه ای که message passing را پیاده کرده این وظیفه را به عهده دارد و طرفین فقط پیام دریافت میکنند یا پیام ارسال میکنند.
اما در روش دوم مدیریت چنین مواردی باعث ایجاد سربار در کتابخانه موردنظر میشود. پس به طور کلی روش اول میتواند سربار کمتری داشته باشد در حالی که استفاده از آن پردردسرتر و سخت تر است.

ب) مزایا و معایب استفاده از هر روش برای ارتباطات بین فرآیندی چیست؟
پاسخ:
مدل حافظه مشترک

مزیت ها	معایب
حداکثر سرعت محاسبات را فراهم می کند زیرا ارتباط از طریق حافظه مشترک انجام می شود، بنابراین فراخوانی سیستم کال ها فقط برای ایجاد حافظه مشترک انجام می شود.	پیاده سازی آن دشوار است زیرا برنامه نویس باید اطمینان حاصل کند که فرآیندها به طور همزمان داده ها را در یک مکان از حافظه نمی نویسند.
برای به اشتراک گذاری حجم زیادی از داده ها مفید است.	برای ارتباط بین فرآیندهایی که در ماشین های جداگانه هستند، مناسب نیست.
از نظر سرعت سریعتر از مدل message passing است.	برای به اشتراک گذاری حجم کم داده مناسب نیست.

مدل انتقال پیام

مزیت ها	معایب
این مدل برای به اشتراک گذاری مقادیر کمی از داده ها مفید است.	وقت گیر است زیرا ارسال پیام از طریق مداخله کرنل (سیستم کال) اجرا می شود.
برای ارتباط بین فرآیندها در ماشین های مختلف مناسب است.	برای به اشتراک گذاری حجم زیاد داده مناسب نیست.
پیاده سازی آن ساده تر از مدل حافظه مشترک است.	از نظر سرعت ارتباط از حافظه مشترک کندتر است.

سوال ۴
مساله ی producer-consumer را به کمک دو روش میتوانیم حل کنیم.
الف)هریک از این دو روش را ذکر کرده و توضیح دهید.
پاسخ :این مساله را به کمک روش message passing و shared memory میتوانیم حل کنیم.

پیاده سازی مربوط به روش shared memory: برای حل این مسله به کمک این روش، متغیرهای زیر در ناحیه ی حافظه ی مشترک بین دوفرایند producer و consumer تعریف میشوند.

```
#define BUFFER_SIZE 10

typedef struct {
    . . .
} item;

item buffer[BUFFER_SIZE];
int in = 0;
int out = 0;
```

- بافر مشترک به صورت یک آرایه ی دایره ای پیاده سازی میشود که شامل ۲پوینتر منطقی است:
- ۱. متغیر in که به خانه ازاد بعدی در حافظه اشاره میکند.
- ۲. متغیر out که به اولین خانه استفاده شده در بافر اشاره میکند.
- زمانی که بافر پرباشد، ((in + 1) % BUFFER SIZE) == out برقرار است.
- زمانی که بافر خالی باشد، in ==out برقرار است.

The producer process using shared memory

```
item next_produced;

while (true) {
     /* produce an item in next_produced */

     while (((in + 1) % BUFFER_SIZE) == out)
        ; /* do nothing */

     buffer[in] = next_produced;
     in = (in + 1) % BUFFER_SIZE;
}
```

The consumer process using shared memory

```
item next_consumed;

while (true) {
     while (in == out)
        ; /* do nothing */

     next_consumed = buffer[out];
     out = (out + 1) % BUFFER_SIZE;

     /* consume the item in next_consumed */
}
```

استفاده از روش message passing برای حل این مساله:

The producer process using shared memory

```
message next_produced;

while (true) {
     /* produce an item in next_produced */

     send(next_produced);
}
```

The consumer process using shared memory

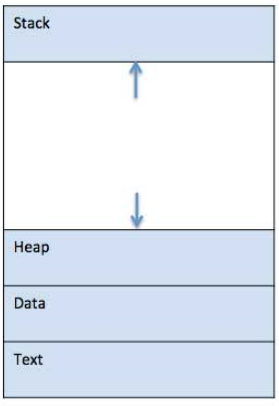
```
message next_consumed;

while (true) {
     receive(next_consumed);

     /* consume the item in next_consumed */
}
```

ب) مزایا و معایب هر یک را بیان کنید.
پاسخ این قسمت در سوال ۳ بخش ب قراردارد.

سوال ۵

اجزای اصلی تشکیل دهنده‌ی یک Process داخل مموری را نام برده و بیان کنید هر بخش چه اطلاعاتی ذخیره میکند.

پاسخ: هنگامی که یک برنامه در حافظه بارگذاری می شود و تبدیل به یک فرآیند می شود، می توان آن را به چهار بخش تقسیم کرد:



Stack	حاوی داده های موقتی مانند پارامترهای تابع، آدرس بازگشتی و متغیرهای محلی است.
Heap	وظیفه ی این بخش، تخصیص حافظه به صورت پویا در طول زمان اجرا به یک فرآیند است.
Text	شامل دستور العمل های ماشین است(باینری کد) که CPU اجرا میکند.
Data	این بخش شامل متغیرهای سراسری و ثابت است.

برای مشاهده ی مقداری که در این بخش ها ذخیره میشود، میتوانیم کد زیر را اجرا کنیم:

```c
// Global variable in the data segment
int globalVar = 42;

int main()
{
    // Local variables in the stack
    int localVar1 = 10;
    int localVar2 = 20;

    // Dynamically allocate memory on the heap
    int *dynamicVar = (int *)malloc(sizeof(int));
    *dynamicVar = 30;

    // Display addresses and values
    printf("Text Segment:\n");
    printf("Code in main(): %p\n", (void *)main);

    printf("\nData Segment:\n");
    printf("Global variable (globalVar): %p, value: %d\n", (void *)&globalVar, globalVar);

    printf("\nHeap:\n");
    printf("Dynamic variable (dynamicVar): %p, value: %d\n", (void *)dynamicVar, *dynamicVar);

    printf("\nStack:\n");
    printf("Local variable 1 (localVar1): %p, value: %d\n", (void *)&localVar1, localVar1);
    printf("Local variable 2 (localVar2): %p, value: %d\n", (void *)&localVar2, localVar2);

    // Free dynamically allocated memory
    free(dynamicVar);
```

خروجی:

```
Text Segment:
Code in main(): 0x80484fb

Data Segment:
Global variable (globalVar): 0x804a02c, value: 42

Heap:
Dynamic variable (dynamicVar): 0x985a008, value: 30

Stack:
Local variable 1 (localVar1): 0xbfe04240, value: 10
Local variable 2 (localVar2): 0xbfe04244, value: 20
```

سوال ۶

اجزای اصلی تشکیل دهندهی Process Control Block را نام برده و بیان کنید هر یک چه اطلاعاتی ذخیره میکند.

Process ID (PID)	یک شناسه منحصر به فرد که توسط سیستم عامل به هر فرآیند اختصاص داده می شود. PID برای تمایز بین فرآیندهای مختلف استفاده می شود.
Process state	وضعیت فعلی فرآیند را نشان میدهد، مانند "در حال اجرا"، "آماده"، "مسدود" و غیره.
Program counter	ثباتی که آدرس دستور بعدی را که باید توسط CPU اجرا شود را نشان می دهد.
CPU Registers	رجیسترهای مختلفی که مقادیر فعلی CPU را ذخیره می کنند، از جمله رجیسترهای همه منظوره، رجیسترهای وضعیت و..
Memory Pointers	اشاره گرهایی که مکان فرآیند را در حافظه نشان می دهند. شامل پوینترهایی به کد فرآیند (بخش متن)، بخش داده، پشته و heap است.
File Descriptors	لیستی از فایل های باز مرتبط با فرآیند. (فایل هایی که از انها مقادیر خوانده میشود یا مقادیر در انها نوشته میشود)
Inter-Process Communication (IPC) Information	شامل جزئیاتی در مورد صف های پیام، سمافورها، حافظه مشترک و ..

سوال۷

پس از اجرای کد زیر، چه تعداد پروسس برحسب n خواهیم داشت؟ توضیح دهید. (با احتساب پروسس والد)

```c
for (i = 0; i < n; i++)
    fork();
```

پاسخ: تابع fork()در یونیکس/لینوکس یک فرآیند جدید را با کپی کردن فرآیند موجود ایجاد می کند. فرآیند جدید که فرزند نامیده می شود، یک کپی دقیق از فرآیند والد است، به جز چند مقدار که تغییر می کنند، مانند شناسه فرآیند.
در این کد ، تابع fork()در حلقه ای فراخوانی می شود که n بار اجرا می شود. هر بار که fork() فراخوانی می شود، یک فرآیند جدید ایجاد می کند.
بنابراین، اگر با یک فرآیند شروع کنیم، پس از اولین تکرار، 2 فرآیند خواهیم داشت.
در تکرار بعدی، هر یک از این 2 فرآیند مجدداً fork() را فراخوانی می‌کنند که در نتیجه 2 فرآیند جدید ایجاد می‌شود، بنابراین در مجموع 4 فرایند خواهیم داشت. این الگو برای n تکرار ادامه می یابد.

بنابراین، تعداد کل فرآیندهای ایجاد شده 2^n خواهد بود که شامل فرآیند اصلی است.
به طور مثال، خروجی اجرای کد برای n=3 به شکل زیر است:
شروع: 1 فرآیند (اصلی)
بعد از اولین تکرار: 2 فرایند (اصلی و 1 فرایند جدید)
بعد از تکرار دوم: 4 فرآیند (اصلی، 1 از تکرار اول، و 2 فرآیند جدید)
پس از تکرار سوم: 8 فرایند (اصلی، 1 از تکرار اول، 2 از تکرار دوم، و 4 فرآیند جدید)
بنابراین، برای n=3، شما با 2^3 = 8 فرآیند مواجه می شوید.

سوال ۸

پس از اجرای برنامه ی زیر، خروجی در Line A چه خواهد بود؟ توضیح دهید.

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

int value = 5;

int main()
{
    pid t pid;
    pid = fork();
    if (pid == 0)
    { /* child process */
        value += 15;
        return 0;
    }
    else if (pid > 0)
    { /* parent process */
        wait(NULL);
        printf("PARENT: value = %d", value); /* LINE A */
        return 0;
    }
}
```

پاسخ: این برنامه با استفاده از fork() یک فرآیند فرزند ایجاد می کند و با افزودن 15 به value مقدار این متغیر را در فرآیند فرزند اپدیت می کند.
فرآیند والد با استفاده از wait(NULL) منتظر می ماند تا فرایند فرزند تکمیل شود و سپس مقدار value را چاپ می کند.
فرایند فرزند یک کپی از حافظه فرایند والد دریافت می کند و هر فرآیند فرزند دارای کپی مخصوص به خود از متغیرها است.
پس مقدار value در فرآیند والد بدون تغییر وهمان 5 باقی می ماند.
بنابراین، خروجی در خط Line A مقدار 5 خواهد بود.

سوال ۹

برنامه زیر را در نظر بگیرید.

```c
int a = 5;
int fd = open(...) //opening a file
int ret = fork();
if(ret >0) {
  close(fd);
  a = 6;
  ...
}
else if(ret==0) {
  printf("a=%d\n", a);
  read(fd, something);
}
```

بعد از اینکه پروسس جدید fork شد، فرض کنید ابتدا پروسس والد قبل از پروسس فرزند زمان‌بندی می‌شود (نوبت اجرا در cpu می‌گیرد). همچنین پروسس فرزند برای اولین بار بعد از اجرای کامل پروسس والد زمان‌بندی می‌شود.

به سؤالات زیر با توضیح مختصر پاسخ دهید:

الف) مقدار a که در پروسس فرزند چاپ می‌شود چند است؟

مقدار a،5 است.

این به این دلیل است که فرآیند فرزند دارای کپی مخصوص به خود از متغیرها است و هر تغییری که در متغیرهای فرآیند والد ایجاد می‌شود روی فرآیند فرزند تأثیری نمی‌گذارد.

ب) آیا تلاش برای خواندن از fd در پروسس فرزند موفقیت‌آمیز خواهد بود؟

هنگامی که یک فرآیند جدید به کمک فورک ایجاد می شود، فرایند فرزند یک کپی از جدول توصیف کننده فایل فرآیند والد را به ارث می برد که شامل توصیفگرهای فایلی است که در فرآیند والد باز بودند. در نتیجه، فرایند فرزند می تواند به توصیفگرهای فایلی که توسط فرآیند والد باز شده است دسترسی داشته باشد و از آن استفاده کند.
ج) با ترتیب زمان‌بندی که بیان شد، آیا پروسس فرزند zombieمی‌شود یا orphan یا هردو؟ با اضافه کردن کد مناسب به این برنامه، از این اتفاق یا اتفاقات جلوگیری کنید.

با ترتیب زمانی ذکر شده، فرایند یتیم می شود.

به این دلیل که والد قبل از فرزند خاتمه می یابد درحالی که فرزند هنوز در حال اجرا است. برای جلوگیری از این اتفاق، می‌توانیم سیستم کال ()wait را اضافه کنیم تا قبل از پایان فرآیند والد، منتظر بمانیم تا فرآیند فرزند تکمیل شود.

```c
int a = 5;
int fd = open(...); // opening a file
int ret = fork();
if (ret > 0)
{
    close(fd);
    a = 6;
    // wait for child process to complete
    wait(NULL);
}
else if (ret == 0)
{
    printf("a = %d\n", a);
    read(fd, something);
}
```

سوال ۱۰. خروجی برنامه ی زیر در Line C و Line P چه خواهد بود؟ توضیح دهید.

```c
#include <pthread.h>
#include <stdio.h>
int value = 0;
void *runner(void *param); /* the thread */

int main(int argc, char *argv[])
{
    pid t pid;
    pthread t tid;
    pthread attr t attr;
    pid = fork();
    if (pid == 0)
    { /* child process */
        pthread attr init(&attr);
        pthread create(&tid, &attr, runner, NULL);
        pthread join(tid, NULL);
        printf("CHILD: value = %d", value); /* LINE C */
    }
    else if (pid > 0)
    { /* parent process */
        wait(NULL);
        printf("PARENT: value = %d", value); /* LINE P */
    }
}

void *runner(void *param)
{
    value = 5;
    pthread exit(0);
}
```

پاسخ: این برنامه با استفاده از ()fork یک فرایند فرزند ایجاد می کند و با استفاده از ()pthread_create یک ترد جدید در فرآیند فرزند ایجاد می کند.

فرایند فرزند با استفاده از () pthread_join منتظر می ماند تا رشته تکمیل شود. thread مقدار value را 5 می کند و با استفاده از ()pthread_exit از آن خارج می شود.

فرآیند والد منتظر می ماند تا فرزند با استفاده از ()waitتکمیل شود و سپس مقدار value را با استفاده از printfچاپ می کند.

از آنجایی که فرآیند فرزند دارای کپی مقدار خود است، مقدار value در فرآیند والد بدون تغییر باقی می‌ماند.

بنابراین، خروجی در خط line P،صفرخواهد بود.
در فرآیند فرزند، thread قبل از خروج مقدار value=5 قرار می دهد. بنابراین، خروجی در خط C، 5خواهد بود.

```c
void makePipeline(char* command1, char** argv1, char* command2, char** argv2){
    int fd[2];
    pipe(fd);
    pid_t pidComm[2];
    pidComm[1] = fork();
    if(pidComm[1] == -1){
        printf("Can't Create New process for pipeline\n");
        return;
    }else if(pidComm[1] == 0){
        close(fd[0]);
        dup2(fd[1], 2);
        dup2(fd[1], STDOUT_FILENO);
        execv(command1, argv1);
        printf("%s: No such command\n", command1);
        exit(0);
    }else if(pidComm[1] > 0){
        pidComm[0] = fork();

        if(pidComm[0] < 0){
            printf("Can't Create New Process for pipeline\n");
            kill(pidComm[1], SIGKILL);
            return;
        }else if(pidComm[0] == 0){
            close(fd[1]);
            dup2(fd[0], 0);
            dup2(fd[0], STDIN_FILENO);
            execv(command2, argv2);
            printf("%s: No such command\n", command2);
            exit(0);
        }else if(pidComm[2] > 0){
            //parent Never use the pipe:
            close(fd[0]);
            close(fd[1]);
            wait(NULL);
            wait(NULL);
        }
    }
    return;
}
```

سوال ۱۱

کد روبرو قسمتی از برنامه shell برای اجرای پایپلاین دو دستور است. (مانند دستور grep new | ls برای لیست‌کردن فایل‌هایی که نامشان شامل new است).

با استفاده از دانش خود و جستجوی دستورات و system callهای ناآشنا توضیح دهید خطوط مشخص‌شده چه می‌کنند و در کل توضیح دهید این کد چگونه اجرای پایپلاین دو دستور را پیاده‌سازی می‌کند.

پاسخ: این کد C یک تابع makePipeline را تعریف می کند که یک خط لوله بین دو دستور ایجاد می کند. خط لوله اجازه می دهد تا از خروجی یک دستور به عنوان ورودی برای دستور دیگر استفاده شود.

این تابع چهار پارامتر دارد: command2 ،argv1 ،command1 و argv2، که در آن command1 و command2 نام دستوراتی هستند که باید اجرا شوند، و argv1 و argv2 آرایه‌هایی از آرگومان‌ها برای هر دستور هستند.

بررسی مرحله به مرحله کد:

۱. (pipe(fd: یک پایپ با دو توصیف کننده فایل (fd [0] برای خواندن، fd[1]برای نوشتن) ایجاد می کند.

۲.(pipComm[1] = fork() : یک فرایند فرزند (pipComm[1]) را برای اجرای اولین فرمان فورک میکند.

۳. فرآیند فرزند(pipComm[1] == 0) :

- انتهای خوانده شده پایپ[0]fd را می بندد.
- به کمک دستور dup2(fd[1],STDOUT_FILENO) ،خروجی استاندارد (STDOUT_FILENO) را به سر نوشتن پایپ (fd[1]) هدایت می کند. به این معنا که از این پس به جای اینکه خروجی پروسس فرزند در خروجی استاندارد نمایش داده شود، در پایپ نوشته می شود. یعنی آنچه را که معمولاً به خروجی استاندارد می‌رود، بردارد و به جای آن وارد لوله شود.
- دستور dup2(fd[1],2)، خطای استاندارد (STDERR_FILENO) را به انتهای نوشتن پایپ (fd[1]) هدایت می کند. این کار تضمین می کند که اگر هر گونه پیغام خطایی از دستور اول وجود داشته باشد، آنها نیز به پایپ می روند.
- خطای استاندارد (file descriptor2): در برنامه های C، خطای استاندارد مانند یک کانال جداگانه است که به طور خاص برای پیام های مربوط به خطاها است.
- سپس اولین دستور را به کمک تابع execv(command1,argv1) اجرا می کند.
- اگر execv ناموفق باشد، یک پیغام خطا چاپ می کند و خارج می شود.

۴. فرآیند والد (pidComm[1]>0):

- فرآیند فرزند دیگری (pidComm[0]) را برای اجرای فرمان دوم فورک می کند.

۵. فرآیند فرزند دوم (pidComm[0] ==0):

- سر مربوط به نوشتن پایپ (fd[1]) را می بندد.
- ورودی استاندارد (STDIN_FILENO) را به سر مربوط به خواندن پایپ [0]fdهدایت می کند. یعنی فرایند فرزندی که قرار است اجرا شود ورودیش را از پایپ برمیدارد.
- دستور دوم execv(command2,argv2) را اجرا می کند.

- اگر execv ناموفق باشد، یک پیغام خطا چاپ می کند و خارج می شود.

۵. فرآیند والد (pidComm[0]>0):

- هر دو انتهای لوله را می بندد، زیرا فرآیند اصلی از آن استفاده نمی کند.
- منتظر می ماند تا هر دو فرایند فرزند به پایان برسند که این کار را با فراخوانی wait(NULL) انجام میدهد.

بررسی توابع خواسته شده:

- pid_t wait(int *status): یک سیستم کال است که به عنوان ارگومان ورودی، یک اشاره گر به یک عدد صحیح که در آن وضعیت خروج از فرآیند فرزند ذخیره می شود، دریافت میکند.
- int dup2(int oldfd,int newfd): یک سیستم کال است که یک توصیفگر فایل موجود را به یک توصیفگر فایل دیگر کپی می کند. در کد، برای تغییر مسیر خروجی استاندارد و خطای استاندارد به انتهای نوشتن لوله استفاده می شود.
- int execv(const char* path, char* const argv[]): یک سیستم کال است که برای جایگزینی فرآیند فعلی با یک فرآیند جدید استفاده می شود. یعنی یک برنامه جدید را از مسیر داده شده بارگذاری و اجرا می کند و آرگومان های ارائه شده را به برنامه جدید ارسال می کند.

موفق باشید :)