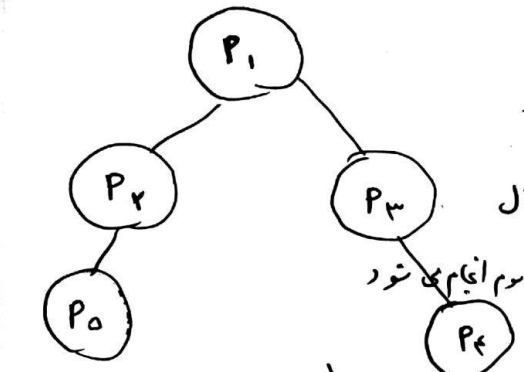
فرهاد امان 9931006

۱-



پس از اجرای خط اول کد ۵ پردازه ایجاد می شود

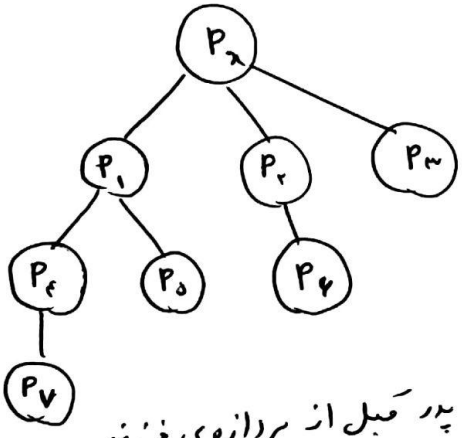
در پردازه ی P_2 به علت ۰ بودن خروجی fork اول

بقیه دستور انجام نشده (به علت وجود and) forks سوم انجام می شود

در پردازه P_1 پس از انجام fork دوم بقیه دستور انجام نمی شود (به علت غیر ۰ بودن خروجی

fork و وجود or) در پردازه P_3 هم fork سوم انجام می شود.

حالا می توانیم به ازای هرکدام از ۵ پردازه ی موجود بقیه میرا بررسی کنیم.



هر پردازه ۷ پردازه ی جدید تولید می کند

پس در نهایت $۸ \times ۵ = \boxed{۴۰}$

پردازه خواهیم داشت.

تمام این محاسبات با این فرض است که پردازه ی پدر قبل از پردازه ی فرزند

terminate نشود چرا که در بعض سیستم عامل ها در این حالت تمام پردازه های فرزندم

terminate خواهند شد.

-۲

CHILD: value = 5

PARENT: value = 0

بعد از اینکه دستور fork انجام می‌شود دو پردازه مجزا ایجاد می‌شوند که تمام فضای حافظه آن‌ها متفاوت و ایزوله از هم است در نتیجه مقدار value در پردازه فرزند هیچ ارتباطی به value در پردازه پدر ندارد و مقدار value در پردازه پدر همان 0 خواهد ماند.

در پردازه فرزند یک thread جدید ایجاد شده و مقدار value را تغییر می‌دهد. توجه کنید که در thread های مختلف یک پردازه متغیرهای global مشترک هستند. در نتیجه با تغییر مقدار value در thread جدید این مقدار در thread اصلی هم تغییر خواهد کرد.

-۳

کمترین مقدار MAX و بیشترین مقدار MAX * 2

دو thread ایجاد کردیم که همزمان اجرا شده و سعی دارند به متغیر مشترک count دسترسی داشته و آن را تغییر دهند.

دستور count = count + 1 در واقع متشکل از چند instruction مجزا است به همین خاطر اجرای همزمان این دستور در دو thread می‌تواند شرایط race condition را ایجاد کند.

به عنوان مثال thread اول این مقدار را برابر 5 می‌خواند و قصد دارد مقدار 6 را در آن بنویسد در این زمان thread دو هم مقدار 5 را میخواند و مقدار 6 و سپس مقدار 6 را میخواند و مقدار 7 را در آن می‌نویسد. سپس thread اول کار نوشتن مقدار 6 را به اتمام می‌رساند و در نهایت مقدار count برابر 6 خواهد بود.

اگر این دو thread در انجام عملیات‌ها با هم هیچ تداخلی نداشته باشند. در نهایت مقدار MAX * 2 در count خواهد بود در صورت تداخل به هر حال حداقل MAX بار عملیات به درستی انجام شده و مقدار count حداقل برابر MAX خواهد بود.

-۴

```c
#include<stdio.h>
#include<linux/types.h>
#include<pthread.h>

int main()
{
    pid_t pid2;
    pid2 = fork();
    if (pid2 == 0) // P2
    {
        printf("P2 Finished\n");
    }
    else if (pid2 > 0)
    {
        pid_t pid3;
```

```
        pid3 = fork();
        if (pid3 == 0) // P3
        {
            pid_t pid4;
            pid4 = fork();
            if (pid4 == 0) // P4
            {
                pid_t pid5;
                pid5 = fork();
                if (pid5 == 0) // P5
                {
                    execlp("/bin/ls", "ls", NULL);
                }
                else if (pid5 > 0) // P4
                {
                    pid_t pid6;
                    pid6 = fork();
                    if (pid6 == 0) // P6
                    {
                        execlp("/bin/sort", "sort", "--version", NULL);
                    }
                    else if (pid6 > 0) // P4
                    {
                        pid_t pid7;
                        pid7 = fork();
                        if (pid7 == 0) // P7
                        {
                            execlp("/bin/find", "find", "--version", NULL);
                        }
                        else if (pid7 > 0) // P4
                        {
                            wait(NULL); // wait for P5
                            wait(NULL); // wait for P6
                            wait(NULL); // wait for P7
                            printf("P4 Finished\n");
                        }
                    }
                }
            }
            else if (pid4 > 0) // P3
            {
                wait(NULL); // wait for P4
                printf("P3 Finished\n");
            }
        }
        else if (pid3 > 0) // P1
        {
            wait(NULL); //wait for P2
            wait(NULL); //wait for P3
            printf("P1 Finished\n");
        }
    }
}
```

۵-
ایجاد یک thread بسیار سبک تر از ایجاد یک process است هنگام ایجاد یک process جدید تمام منابع باید دوباره ایجاد شوند. از جمله register ها، program counter، حافظه‌های heap و stack، تمام متغیر های global، تمام کد برنامه یا همان text و به طور کلی تمام منابع هنگام ایجاد یک thread جدید تنها منابع program counter ،stack و register ها جدا هستند و در بقیه منابع process بین thread های یک برنامه مشترک خواهند بود به همین دلیل ایجاد یک thread بسیار سبک تر است چون نیازی به ایجاد دوباره منابعی مانند heap یا کد های برنامه نیست.

۶-
هنگامی که عمل context switch در thread ها در حال انجام است. بسیاری از منابع نیازی به ذخیره شدن و بازیابی دوباره ندارند. و تنها مواردی مانند program counter , stack pointers و register ها نیاز به ذخیره شدن و بازیابی مجدد دارند و نیازی به تغییر address space نیست. اما در هنگام context switch در process ها در واقع تمام PCB باید ذخیره و بازیابی شود که علاوه بر موارد موجود در thread ها موارد دیگری از جمله memory adress ها , page table ها و همچنین kernel resources را هم شامل می‌شود. و در واقع address sapce به طور کلی تغییر می‌کند به همین خاطر این عمل بسیار پر هزینه تر از context switch در thread ها است.