دانشگاه صنعتی اصفهان

دانشکده مهندسی برق و کامپیوتر

عنوان: تکلیف سوم درس سیستم‌های عامل ۱

نام و نام خانوادگی: علیرضا ابره فروش

شماره دانشجویی: ۹۸۱۶۶۰۳

نیم سال تحصیلی: پاییز ۱۴۰۰

مدرّس: دکتر محمّدرضا حیدرپور

دستیاران آموزشی: مجید فرهادی - دانیال مهرآیین - محمّد نعیمی

فهرست مطالب

۱ سوال اول

۱.۱ آ

Table 1:

	Process	Thread
1.	Process means any program is in execution.	Thread means segment of a process.
2.	Process takes more time to terminate.	Thread takes less time to terminate.
3.	It takes more time for creation.	It takes less time for creation.
4.	It also takes more time for context switching.	It takes less time for context switching.
5.	Process is less efficient in term of communication.	Thread is more efficient in term of communication.
6.	Multi programming holds the concepts of multi process.	We don’t need multi programs in action for multiple threads because a single process consists of multiple threads.
7.	Process is isolated.	Threads share memory.
8.	Process is called heavy weight process.	A Thread is lightweight as each thread in a process shares code, data and resources.
9.	Process switching uses interface in operating system.	Thread switching does not require to call a operating system and cause an interrupt to the kernel.
10.	If one process is blocked then it will not effect the execution of other process	Second thread in the same task could not run, while one server thread is blocked.
11.	Process has its own Process Control Block, Stack and Address Space.	Thread has Parents’ PCB, its own Thread Control Block and Stack and common Address space.
12.	If one process is blocked, then no other process can execute until the first process is unblocked.	While one thread is blocked and waiting, a second thread in the same task can run.
13.	Changes to the parent process does not affect child processes.	Since all threads of the same process share address space and other resources so any changes to the main thread may affect the behavior of the other threads of the process.

۲.۱ ب

- register setِthreadها بین threadها غیر مشترک است. این رجیسترها وضعیت پردازنده را مشخص می‌کنند و threadها عملیات محاسباتی مستقل از یکدیگر دارند بنابراین virtual registerهای نمیتوانند مشترک باشند. مثلا Program Counter(PC) و Stack Pointer(SP) در هر thread مجزا و مخصوص به همان thread است.
- TIDِ هر thread یکتا و مختص همان thread است. در واقع داخل یک پروسس، TID وجه تمییز بین threadهای یک پروسس است. پس نمی‌تواند مشترک باشد.
- stackِ هر thread، متغیرهای محلی و آدرس بازگشتی را به صورت مجزا ذخیره می‌کنند.

۳.۱ ج

برای مدلسازی بسیاری از مسائل موجود در دنیای واقعی که به نحوی چند کار باهم همزمانی دارند به برنامه‌های concurrent نیاز داریم و نمی‌توانیم تنها به متدهای برنامه‌نویسیِ خطی و ترتیبی اتکا کنیم. برنامه‌نویسی چندنخی در این زمینه به ما کمک می‌کند، به طوریکه به هر یک از کارهای همزمان یک thread نظیر می‌کنیم و این تعاملِ threadها باهم روند یک کار را به طور دقیق مدل می‌کند.

۴.۱ د

در پردازنده‌های تک هسته‌ای هنگامی که یک thread منتظر I/O سنکرون است، scheduler، threadهای دیگر را(برای مثال read()) schedule می‌کند و هدر رفت زمان کمتر رخ می‌دهد. از دیگر مزایای برنامه‌نویسی چندنخی در پردازنده‌های تک هسته‌ای عدم جابه‌جایی ترتیب اجرای دستورات در بسیاری از پیاده‌سازی‌ها است. مشکل data race نیز به دلیل عدم وجود پیچیدگی در ترتیب اجرای دستورات، رفع می‌شود.

۵.۱ ه

جججججججججججججججج

۶.۱ و

Ticketlockها بر خلاف Spinlockها مشکل گرسنگی مفرط ندارند. Fetch and Add نیز دچار گرسنگی مفرط نمی‌شود.

۷.۱ ز

۴ منبع و ۳ رشته داریم. درنتیجه حداقل یکی از رشته‌ها حداقل ۲ تا از منابع را اشغال می‌کند و circular wait رخ نمی‌دهد. پس Deadlock نمی‌تواند رخ دهد.

۲ سوال دوم

۱.۲ طول بافر محدود

```
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
#include <stdlib.h>

sem_t unocc; //occupied slots
sem_t occ; //unoccupied slots

void* produce(void* arg)
{
        while(1)
        {
                sem_wait(&unocc);
        sleep(rand() % 100 * 0.01);
        sem_post(&occ);
        sleep(rand() % 100 * 0.01);
        }
}

void* consume(void* arg)
{
        while(1)
        {
                sem_wait(&occ);
        sleep(rand() % 100 * 0.01);
        sem_post(&unocc);
        sleep(rand() % 100 * 0.01);
        }
}

int main(int argv, char* argc[])
```

```
{
        int size = 0;
        printf("Enter buffer size: ");
        scanf("%d", &size);
        pthread_t producer, consumer;
        pthread_attr_t a1;
        sem_init(&occ, 0, 0);
        sem_init(&unocc, 0, size);
        pthread_attr_init(&a1);
        pthread_attr_setdetachstate(&a1, PTHREAD_CREATE_JOINABLE);
        if(pthread_create(&producer, &a1, produce, 0))
        {
                printf("Failed to create producer thread!\n");
                exit(-1);
        }
        if(pthread_create(&consumer, &a1, consume, 0))
        {
                printf("Failed to create consumer thread!\n");
                exit(-1);
        }
        pthread_attr_destroy(&a1);
        if(pthread_join(producer, 0))
        {
                printf("Failed to join producer thread!\n");
        }
        if(pthread_join(consumer, 0))
        {
                printf("Failed to join consumer thread!\n");
        }
        pthread_exit(0);

        return 0;
}
```

۲.۲ طول بافر بی‌نهایت

```
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
#include <stdlib.h>

sem_t occ; //occupied slots

void* produce(void* arg)
{
        while(1)
        {
        sem_post(&occ);
        sleep(rand() % 100 * 0.01);
        }
}

void* consume(void* arg)
{
        while(1)
        {
                sem_wait(&occ);
        sleep(rand() % 100 * 0.01);
        }
}

int main(int argv, char* argc[])
{
        pthread_t producer, consumer;
        pthread_attr_t a1;
        sem_init(&occ, 0, 0);
        pthread_attr_init(&a1);
        pthread_attr_setdetachstate(&a1, PTHREAD_CREATE_JOINABLE);
        if(pthread_create(&producer, &a1, produce, 0))
```

```
        {
                printf("Failed to create producer thread!\n");
                exit(-1);
        }
        if(pthread_create(&consumer, &a1, consume, 0))
        {
                printf("Failed to create consumer thread!\n");
                exit(-1);
        }
        pthread_attr_destroy(&a1);
        if(pthread_join(producer, 0))
        {
                printf("Failed to join producer thread!\n");
        }
        if(pthread_join(consumer, 0))
        {
                printf("Failed to join consumer thread!\n");
        }
        pthread_exit(0);
        return 0;
}
```

۳ سوال سوم

ابتدا بررسی می‌کنیم که TLB Miss رخ داده است یا TLB Hit. این کار در ۱ نانو ثانیه انجام می‌شود. سپس دو حالت زیر را در نظر می‌گیریم:

- TLB Miss رخ دهد. در این حالت داریم: $P_{TLBMiss}(2 * T_{DRAM} + P_{PageFault} * T_{Disk}) = 4ns$
- TLB Miss رخ ندهد. در این حالت داریم: $(1 - P_{TLBMiss}) * (T_{Cache} + P_{CacheMiss} * P_{PageFault} * T_{Disk}) = 2.97ns$

در نهایت میانگین زمان دسترسی به صفحات حافظه برابر است با: $1ns + 4ns + 2.97ns = 7.97ns$

۴ سوال چهارم

۱.۴ الف

گام اول: تجزیه‌ی آدرس مجازی به Seg، VPN، و Offset:
۲ بیت پرارزش مربوط به سگمنت است، چون صفحه‌ها ۳۲ بایتی هستند پس ۵ بیت برای نشان دادن آن‌ها نیاز داریم که از راست برای آفست جدا می‌کنیم و ۵ بیت باقی‌مانده برای VPN است.

0x45d = 01 00010 11101

SN = 01

VPN = 00010 = 2

Offset = 11101 = 29

Address of PTE = Base[SN] + VPN*sizeof(PTE) = 64 + 2*1 = 66

بایت ۶۶ در صفحه‌ی با PFN = 2 قرار دارد و بایت سوم آن است.

Page with PFN = 2: c3

c3 = 1 1000011 = 67

valid bit = 1

PFN = 67

آفست ۲۹ در صفحه‌ی ۶۷ که آدرس ۴۰ است. پس برای ترجمه آدرس مجازی به آدرس فیزیکی، به صفحات ۲ و ۶۷ رجوع می‌شود.
بله چون بیتِ valid یک است و بایت مورد نظر c3 است که در صفحه‌ی ۲ قرار دارد.

۲.۴ ب

گام اول: تجزیه‌ی آدرس مجازی به Seg، VPN، و Offset:
۲ بیت پرارزش مربوط به سگمنت است، چون صفحه‌ها ۳۲ بایتی هستند پس ۵ بیت برای نشان دادن آن‌ها نیاز داریم که از راست برای آفست جدا می‌کنیم و ۵ بیت باقی‌مانده برای VPN است.

0xc85 = 11 00100 00101

SN = 11

VPN = 00100 = 4

Offset = 00101 = 5

Address of PTE = Base[SN] + VPN*sizeof(PTE) = 640 + 4*1 = 644

بایت ۶۴۴ در صفحه‌ی با PFN = 20 قرار دارد و بایت پنجم آن است.

Page with PFN = 20: 20

20 = 0 0100000

valid bit = 0

خیر چون بیتِ valid صفر است، این آدرس مجازی به این پروسس تخصیص داده نشده است و تنها به صفحه‌ی ۲۰ رجوع می‌شود.

منابع

[1] https://gateoverflow.in/150841/Tlb-and-page-fault