70088 ExerciseTypes.CW1

CW

Submitters

---

sf23 Shihan Fu

Emarking

Computer Systems (70088) – Coursework

Q1. Give an example of when a process switches from user to kernel mode and then back to user mode. (1 mark)

This can be achieved with system calls, in cases such as accessing hardware devices or modifying system settings. One example could be that user writes codes to read from a file. When the main program is executed, the processor is in the user mode. When it executes the read function, it triggers the transaction of modes from user mode to kernel mode to execute the system call and to perform operations to read from file. Once the kernel is done with the job, it then switches back to user mode maybe with a return value.

Q2. Which of the following is not an operating system?

(a) Unix (b) Linux (c) Ubuntu (d) Mac OS X (e) Android (f) Windows 10

Answer: I think all of them can be called operating systems. If I must choose one, then according to the strict definition of OS, I will choose (b) because it is a family of all Linux operating systems.

Solution:

foundation of other OS: (a)

Desktop/ Laptop: (b), (c), (d), (f)

Server OS: (b)

Smartphones: (e)

Q3. The FCFS scheduling algorithm works very well for systems with I/O usage.

(a) True (b) False (1 mark)

Answer: b. False

Solution: FCFS is not used in I/O usage because there may be a case where a long process arrives first. Then the CPU will be occupied for a long time, which we do not want it to happen.

Q4. What is the average turnaround time of 4 processes (execution times of 3, 4, 2, and 5s) using the Shortest Job First scheduling algorithm? (a) 9 (b) 6.5 (c) 7.5 (d) 8.25 (1 mark)

Answer: c. 7.5

Solution: For Shortest Job First scheduling algorithm, the processes should be in the order of 2, 3, 4, 5. Then the average turnaround time is $\frac{2+(2+3)+(2+3+4)+(2+3+4+5)}{4} = 7.5$

Q5. Which of the following instructions should only be allowed in kernel mode?

(a) Disabling all interrupts

(b) Reading the time of day clock

(c) Reading the /proc file system

(d) Running the top command

(e) Killing a user process

(f) Sending a signal to another process (1 mark)

Answer: a.

Solution: In kernel mode, the program can access all memory and handle interrupts. In user

mode, it can access limited memory and cannot handle interrupt directly.

Q6. Which of the following approaches do not require any knowledge of the system state?
(a) Deadlock avoidance
(b) Deadlock detection and recovery
(c) Deadlock prevention
(d) None of the above (1 mark)
Answer: c.

Q7. Which of the following are true?
(a) Interactive systems use non-preemptive scheduling
(b) Turnaround times in preemptive systems are more predictable
(c) A weakness of priority scheduling is priorities may not be meaningful
(d) If all processes are I/O-bound, ready queue will almost always be full
(e) Preemptive scheduling suspends a running process before its time slice expires (2 marks)
Answer: c, e
a. They use preemptive scheduling to guarantee a fast response to new requests.
b. Turnaround time is less predictable in preemptive system because the preemptive orders and time cannot be guaranteed by the system.
d. If all processes are I/O bound, the ready queue will almost always be empty and the short-scheduler will have little to do. If all processes are CPU bound, the I/O waiting queue will almost always be empty, devices will go unused, and the system will be unbalanced. (Jordan University of science and technology)

Q8. How many child processes are created? (a) 1 (b) 3 (c) 4 (d) 5 (e) 7 (f) 10
Answer: e.
Solution:

```
for       i=0       1         2
            father    father    father
                                  son
                        son     father
                                  son
             son      father    father
                                  son
                        son     father
                                  son
```

9.

```
/*
 * @Author: shihan
 * @Date: 2023-11-29 19:19:50
 * @version: 1.0
 * @description: write a producer-consumer codes
 */
```

```cpp
#include <iostream>
#include <thread>
#include <semaphore.h>
#include<pthread.h>
#include <unistd.h> // sleep
#include <chrono>
#include <mutex>
#include <condition_variable>

#define MAX_QUEUE_SIZE 100

using namespace std;

sem_t qmutex, space, item;
mutex queueMutex;
condition_variable queueCondition;

class myQueue{
    public:
        int capacity;
        int *queue;
        myQueue(int _capacity):capacity(_capacity){
            queue = new int[capacity];
            front = 0;
            tail = 0;
        }
        int front;
        int tail;

        void add_item(int item){

            if(*(queue+tail)==0){
                // could add
                *(queue+tail)=item;
                tail = (tail+1)%capacity;
            }
            else{
                //could not add
                cout << "full" << endl;
                exit(1);
            }
        }

        int pop_item(){
```

sub check? Semaphore should do it.

```
        int num = *(queue+front);
        cout << "pop from queue : " << *(queue+front) << "!" << endl;
        *(queue+front) = 0;
        front = (front+1)%capacity;
        check_empty();
        return num;
    }

    bool check_empty(){
        if(*(queue+front)==0){
            cout << "the queue is empty" << endl;
            return true;
        }
        cout << "the queue is not empty" << endl;
        return false;
    }
};

myQueue q = myQueue(MAX_QUEUE_SIZE);

void *producer(void *a){
    int* nums = static_cast<int*>(a);
    // Print the value at the memory location pointed to by (charPtr + 1)
    int num = nums[0];
    for(int i=0;i<num;i++){
        // produce item
        int job = rand() % 10 + 1;  // Random integer between 1 and 10
        cout << "generate job of : " << job << endl;
        timespec ts;
        clock_gettime(CLOCK_REALTIME, &ts);
        ts.tv_sec += 10;  // set timeout to be 10 seconds
        if (sem_timedwait(&space, &ts) == -1) {
            if (errno == ETIMEDOUT) {
                cout << "timeout, producer exit" << endl;
                pthread_exit(0);
            } else {
                cout << "other error, producer exit" << endl;;
                pthread_exit(0);
            }
        }
        // sem_wait(&qmutex);
        cout << "producer produces a job to queue" << endl;
        lock_guard<mutex> lock(queueMutex);
        q.add_item(job);
```

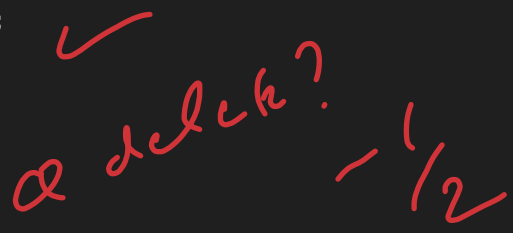
Same as calling $-\frac{1}{2}$

Should be for condition $-\frac{1}{2}$

```
        // sem_post(&qmutex);
        sem_post(&item);
        queueCondition.notify_one();

    }
    cout << "producer thread exit" << endl;
    pthread_exit(0);
}

void *consumer( void *b){

    while(1){
        unique_lock<std::mutex> lock(queueMutex);
        if (queueCondition.wait_for(lock, std::chrono::seconds(10), [&]
{ return !q.check_empty(); })) {
            sem_wait(&item);
            // sem_wait(&qmutex);
            cout << "consumer try to consume a job from queue" << endl;
            lock.unlock();
            int sleep_time = q.pop_item();
            cout << "consumer will sleep for "<< sleep_time << " seconds." <<
endl;
            sleep(sleep_time);
            // std::this_thread::sleep_for(std::chrono::seconds(sleep_time));

            // sem_post(&qmutex);
            sem_post(&space);
        }
        else {
            // If no jobs left and no new jobs added during the wait, exit
            std::cout << "No new jobs. Exiting." << std::endl;
            break;
        }
    }


    pthread_exit(0);

}


int main(){

    cout << "--------------------------" << endl;
    cout << "please put in the queueSize: "<< endl;
```

```
    int queueSize = 5;
    cin >> queueSize;

    cout << "please put in the jobsPerProducer: "<< endl;
    int jobsPerProducer =3;
    cin >> jobsPerProducer;

    cout << "please put in the numProducers: "<< endl;
    int numProducers = 3;
    cin >> numProducers;

    cout << "please put in the numConsumers: "<< endl;
    int numConsumers = 3;
    cin >> numConsumers;

    if (queueSize <= 0 || jobsPerProducer <= 0 || numProducers <= 0 ||
numConsumers <= 0) {
        cerr << "Invalid input values. Please provide positive integers.\n";
        return 1;
    }
    if(queueSize >= MAX_QUEUE_SIZE){
        cerr << "Invalid queue size: out of max queue size 100!" << endl;
        return 1;
    }
    // initialize the semaphares
    sem_init(&qmutex, 0, 1);
    sem_init(&space,0,queueSize);
    sem_init(&item, 0, 0);

    // initiate the circle queue
    myQueue q = myQueue(queueSize);

    // create a threadpool
    pthread_t threadPool[numProducers+numConsumers];

    // set the job per producer as array
    int numbers[] = {5};
    numbers[0] = jobsPerProducer;

    // create the threads for producers
    for(int i=0;i<numProducers;i++){
        pthread_t temp;
        if(pthread_create(&temp,NULL,producer,numbers)==-1){
            cout << "fail to create a producer" << endl;
```

```
            exit(1);
        }
        cout << "create a producer!"<< endl;
        threadPool[i]=temp;
    }

    // create the threads for consumers
    for(int i=numProducers;i<numProducers+numConsumers;i++){
        pthread_t temp;
        if(pthread_create(&temp,NULL,consumer,NULL)==-1){
            cout << "fail to create a consumer" << endl;
            exit(1);
        }
        cout << "create a consumer!"<< endl;
        threadPool[i]=temp;
    }

    // execute threads
    void *result;
    for(int i=0;i<numProducers+numConsumers;i++)
    {
        if(pthread_join(threadPool[i],&result)==-1)
        {
            printf("fail to recollect\n");
            exit(1);
        }
    }

    // cancel the threads
    for(int i=0;i<numProducers+numConsumers;i++)
    {
        pthread_cancel(threadPool[i]);
    }

    // destroy the sem
    sem_destroy(&qmutex);
    sem_destroy(&space);
    sem_destroy(&item);

    return 0;

}
```



sample output:

```
---------------------------
please put in the queueSize:
5
please put in the jobsPerProducer:
3
please put in the numProducers:
1
please put in the numConsumers:
3
create a producer!
create a consumer!
create a consumer!
create a consumer!
generate job of : 4
producer produces a job to queue
the queue is empty
the queue is empty
generate job of : 7
producer produces a job to queue
the queue is not empty
consumer try to consume a job from queue
pop from queue : 4!
the queue is empty
consumer will sleep for 4 seconds.
generate job of : 8
producer produces a job to queue
producer thread exit
the queue is not empty
consumer try to consume a job from queue
pop from queue : 7!
the queue is not empty
consumer will sleep for 7 seconds.
the queue is not empty
consumer try to consume a job from queue
pop from queue : 8!
the queue is empty
consumer will sleep for 8 seconds.
the queue is empty
the queue is empty
the queue is empty
the queue is empty
No new jobs. Exiting.
the queue is empty
```

No new jobs. Exiting.
the queue is empty
No new jobs. Exiting.