University of Dhaka

Department of Computer Science and Engineering

3rd Year 2nd Semester B.Sc. Final Examination, 2018

CSE-3201: Operating Systems

Total Marks: 60 **Time: 3 Hours**

(Answer any four (4) of the following Questions)

1. a) In an attempt to become known as the next Linus Torvalds, Prof. F. C. Coder, Ph.D. intends to modify the standard Unix scheduler. Instead of the complicated multi-level priority queue scheme implemented by most versions of Unix, Prof. Coder proposes using non-preemptive Shortest Job First (SJF) scheduling.

Process	Arrival time	Total CPU burst	I/O bound job (when, length)
P_1	5	200	(25,200),(75,300),(150,100),(175,50)
P_2	0	300	(75,130),(125,230),(225,120),(275,150)
P_3	0	400	(125,160),(175,250),(250,200),(375,150)
P_4	15	100	(35,100),(110,230)
P_5	0	150	(25,120),(50,100),(130,100)

(i) Assuming a correct and successful implementation, calculate the average waiting time and turn-around time for the following processes consider time unit in nanosecond. [6]

(ii) Assuming a correct and successful implementation, what impact on the worst-case maximum waiting time and speed-up would you expect Prof. Coder's modified scheduler to have? [4]

b) In an attempt to become known as the next Bill Gates, Prof. F. C. Coder, Ph.D. has built yet another scheduler for interactive multitasking operating systems. Intended for modest desktop computers running at about 100 million instructions per second (10 MIPS), the schedule is based on a single-queue preemptive round-robin scheme with a quantum of 100 nanoseconds (10/1,000,000,000 of a second). Discuss the performance of this schedule, especially concerning efficiency and response time. [5]

2. Prof. F. C. Coder, Ph.D. has been hired by a bank to design and implement an account management database system. The system is to keep track of each account's balance and allow transfers between them. For reasons known only to himself, Prof. Coder has decided to implement the system entirely in the memory of a single process, with all the authorized users running simultaneous threads with access to the shared memory account database. The account database consists of an array of individual account structures. These structures contain a flag indicating whether the account is valid, its account balance (an integer - this bank deals only in whole dollars), and a binary semaphore that can be used to lock the account during transactions:

```
1 struct account  {
2    int valid; /* account validity flag - 0 = invalid , 1 = valid */
3    int balance; /* balance in dollars */
4    SEMAPHORE s; /* account lock */
5 } accounts [NACCOUNTS];
```

Account numbers are simply integer indices into the accounts array. For example, the transfer function moves money between two accounts as follows:

```
1  int transfer ( int amount, int from, int to )
2  {
3  /* check that account is valid */
4  if ( ( from < 0) || ( from >= NACCOUNTS) ||
5  ( to < 0) || ( to >= NACCOUNTS) )
6  return INVALIDACCOUNT;
7  if ( amount < 0)
8  return INVALIDAMOUNT;
9  lock ( from , to ); /* lock from and to ( prevent race condition ) */
10 if ( !accounts [ from ] . valid || !accounts [ to ] . valid ) {
11 unlock ( from , to );
12 return INVALIDACCOUNT;
13 }
14 accounts [ from ] . balance = account [ from ] . balance - amount;
15 accounts [ to ] . balance = account [ to ] . balance + amount;
16 unlock ( from , to ); /* unlock them */
17 return OK;
18 }
```

Fortunately, the system has standard DOWN (P), and UP (V) semaphore system call... Unfortunately, about 20 minutes before the system was to go online, Prof. Coder left town, having finished everything except the lock and unlock functions. The bank manager is naturally very unhappy and was last heard muttering, "When they find out I hired that Coder idiot, I'll be as dead as Sweetie π".

a) Implement lock and unlock to allow exclusive access to two accounts which prevent deadlocks. Assume, standard DOWN and UP semaphore functions, that the semaphores in the accounts array have been initialized correctly (although no other semaphores are available), that only your functions will be used to operate the semaphores, and that each thread will call your lock function exactly once for each transaction before calling unlock. You may also assume that all appropriate validation and bounds checking has been performed before your functions are called (e.g., the from-to arguments will always represent valid entries in the accounts array). [4]

b) Make a reasonable argument for why the use of your functions will prevent deadlock (e.g., explain the technique you are using and the conditions that can or cannot occur with this technique). [3]

c) An operating system contains three resource classes, namely R_1, R_2 and R_3. The number of resource units in these classes is 7, 7, and 10 respectively. The current resource allocation state is as shown below [8]

Process	Allocated Resources				Maximum Need		
	R_1	R_2	R_3		R_1	R_2	R_3
P_1	2	2	3	P_1	3	6	8
P_2	2	0	3	P_2	4	3	3
P_3	1	2	4	P_3	3	4	4

(i) Is the current allocation state safe?

(ii) Would the following requests be granted in the current state? Process P_1 requests(1, 1, 0), Process P_3 requests(0, 1, 0) (3), and Process P_2 requests(0, 1, 0)

3. a) Explain the difference between internal and external fragmentation. [2]
b) Compare the main memory organization schemes of contiguous memory allocation, pure segmentation and pure paging with respect to the following issues: [5]
(i) External fragmentation
(ii) Internal fragmentation
(iii) Ability to share code across processes
c) Consider the paging system with the page table stored in the memory. [3]
(i) If a memory reference takes 200 ns, how long does a paged memory reference take?
(ii) If we add TLBs and 75% of all page-table references are found in the TLBs, what is the effective memory reference time (Assume that accessing TLB takes 0 ns, if the entry is there)?
d) Compare the segmented paging scheme with the hashed page table scheme for handling large address space. Under what circumstances is one scheme preferable to the other? [3]
e) Discuss the hardware support required to support demand paging. [2]

4. a) You are one of the design team of the new Xinu file system. As part of this design, you have proposed an inode structure similar to that in "traditional" Unix. (Blocks are assumed to be 4KB, and file pointers are 4 bytes.) [5]
(i) Give details of your proposal for the inode structure.
(ii) Calculate the maximum file size, in blocks.
b) Several modern file systems make use of journaling. Describe how a journal is used by the file system and the situations in which it is beneficial. [5]
c) Modern file system (with vfs concept) has support for the native file system as well as a remote file system. Explain the necessary steps concerning the system call mechanism (based on RS3000) to open and read a file from a network file system (NFS). [5]

5. a) Consider the following page reference string 1, 9, 5, 7, 9, 3, 5, 2, 9, 4, 2, 5, 6, 9, 2, 3, 5, 6, 7. How many page faults would occur for the following page replacement algorithms, assuming an allocation of 4 frames? [5]
(i) LRU with 1-bit counter
(ii) FIFO
(iii) Optimal

Suppose the page table for the currently executing process is as follows. All values decimal, indices are numbered from zero. Addresses are memory bytes addresses. The address offset is 10 bits. Describe precisely how, in general, the virtual address generated by the CPU is translated into a physical address, with the aid of a diagram. To what physical address (if any) would the following virtual address correspond? (i) 1056, (ii) 2321, (iii) 5099, (iv) 4235, (v) 309, (vi) 10099 [6]

Virtual Page#	Valid Bit	Reference bit	Modify bit	Page frame#
0	1	1	0	4
1	1	1	1	-
2	0	0	0	5
3	1	0	0	2
4	0	0	0	-
5	1	0	1	10

c) If Mr. X., a system manager of the cybernetic system wants to see active maximum 15,000 processes running on a system. What should be done if the system goes to thrashing? Why? [4]

6. a) Which disk-arm scheduling algorithm will show the best performance when a request of a segment is completed at once? Assume, the segment is a collection of disk block access requests (in order of arrival time). Why? [5]

b) Suppose a disk has 5000 track and 300 sectors in each track. Maximum seek time 10ms, and on track latency is 1ms. Determine the average waiting time of a process for the following requests of disk block using elevator algorithm and SSTF. The current head position is 654, and the previous head position was 233 track (requests placed at time 0) 240, 223, 453, 76, 890, 20, 67, 35, 96, 3176, 2330, 767, 532, 785, 4999, 2356, 355, 0. [6]

c) What is the content of superblock? How i-nodes are distributed in the file system to minimize the access time? Why modern file system (such as etx2 or ext3) maintains multiple copies of the same superblock? [4]

University of Dhaka
Computer Science and Engineering
Mid Term, Third Year, Second Semester
CSE3201– Operating System, 90+ minutes

Prof. Dr. MHK Tushar

2018/10/04

This exam contains 2 pages and 5 questions. **Total of points is 50 (will be converted to 20).** Good luck and Happy day!

---

1. (6 points) Suppose a kernel library function has starting address at **0x3F6F122A** (which is in kernel address spaces). How can this library function be used (or called) by the user program? What will happen to c0_status register before and after the execution the function?

2. (16 points) Determine the critical section (indicate from and to line numbers) from the following segment of code (in page 2) which counts the frequency of numbers appears in the random selection. Modify the code to solve the critical section problem. Your solution must satisfy four necessary conditions and free from busy waiting and maximum parallelism.

3. (6 points) Determine the effective access time for 85% cache hit ratio. Assume memory and cache access times are 10ns and 1ns accordingly.

4. *(6 points) Why thread switching requires less overhead or time compared to process switching? How is trapframe affected by context-switching?*

5. *(16 points) For the following* snapshot of a system, determine whether the system is in safe-state. *If the requests are granted as in the given sequence then what happens? Draw a resource allocation graph if the system is in deadlock* and mark the cycle. What should be the correct sequence of *accepting requests to guarantee that* the system never enters deadlock? Where total resource *matrix is E:[8 6 7 5 8]*, and available before taking any request matrix is **A:[8 6 7 5 8]**. Assume 'release [x]' releases all resources allocated to process 'x' and the process will not request further. Your solution must respect and process all the requests. Moreover, symbols P: process id, ReqSeq: Request the resource instances, and Rn: Request number.

Rn	P:[ReqSeq]	Rn	P:[ReqSeq]	Maximum Need	P
01	2:[1 0 0 0 0]	10	1:[0 0 1 0 0]	8 5 3 5 6	0
02	1:[0 0 1 2 0]	11	3:[1 0 0 1 0]	3 2 7 5 4	1
03	2:[1 0 0 0 0]	12	0:[4 1 1 1 0]	7 1 2 3 1	2
04	4:[3 2 1 3 5]	13	0:[2 1 1 0 0]	4 3 6 5 7	3
05	2:[3 0 0 0 0]	14	2:[2 1 2 3 1]	3 3 7 3 8	4
06	1:[1 2 1 2 3]	15	4:[0 0 0 0 1]		
07	1:[2 0 4 1 1]	16	0:[2 3 1 4 6]		
08	4:[0 0 0 0 1]	17	4:[0 1 6 0 1]		
09	3:[2 3 3 1 2]	18	3:[1 0 3 3 5]		

```
1  #define n_thread 50 //number of threads
2  #define limit_val 1000000000 //Number of times run the "rand()" function
3
4  sem_t finished; //declare semaphore
5
6  static std::vector<int> frequency(100,0); //storage for the frequency of the
       number
7  static std::vector<int> thread_n(50,0); //number of valid choices
8  static int total_count=0;
9
10 void P(sem_t *sem_x){sem_wait(sem_x);} //down function
11
12 void V(sem_t *sem_x){sem_post(sem_x);} //up function
13
14 //Number frequency : effeciency of C++ rand function
15 void frequencyOfNumber(int thd){
16     srand (time(NULL)); //randomize
17     bool flag=0;
18     int x=rand()%100;//choose a number between 0 to 99
19     int y=frequency[x]; //retrive the frequency of x
20     std::cout<<thd<<" chooses number "<<y<<std::endl;
21     y=y+1;
22     if(y!=frequency[x]+1){ //check if the increament is valid
23         std::cerr<<"Sometime wrong!!"<<thd<<" number "<<y<<" !="<<frequency[x]<<
               "+1"<<std::endl;
24     }else{
25         flag=1; //increament is valid
26     }
27     frequency[x]=y; //store current value of y
28     if(flag){
29         total_count+=1; //total number of valid increment
30         thread_n[thd]+=1; //record the thread 'thd' increments
31     }
32 }
33 //Thread function
34 void *thread_counter(void *thd){
35     while(true){
36         int n=total_count;
37         if(n==limit_val){break;}
38         frequencyOfNumber((int)thd);
39     }
40     V(&finished); //job finished!!
41 }
42 //main function : start execution from here!!
43 int main(){
44     sem_init(&finished,0,0); //initialized shemaphore
45     pthread_t my_thread[n_thread]; //thread names or id
46     for(int t=0;t<n_thread;t++){ //create n_thread threads
47         //pthread_create(thread_name, thread_attr, thread_func, thread_args)
48         pthread_create(&my_thread[t],NULL,&thread_counter,(void*)t);
49     }
50     for(int i=0;i<n_thread;i++){
51         P(&finished); //all done
52     }
53 }
```