CS 3100: Operating Systems and Concurrency Spring 2024
Take-home Final Exam

Total Marks: 53

Name: Paige Davidson

A#: A02339425

Carefully Read the following Instructions at least two times:

- Write down your Name and A-number on top of the answer script.

- Clearly mark your answers, where **Ans-i** will be considered as the answer of **Q-i** (i = 1, 2, ..., 10). An answer that is not clearly marked cannot be graded.
- Your answer should be typed in font size: 11.
- The answer script needs to be submitted on Canvas as a single pdf file. Do Not email your submission to Instructor/TA – it is not acceptable.
- Deadline for submission: **4:50 PM on Thursday, April 25**. If your submission is turned in after that time, even just 1 second after, will be considered late. **No late submission will be accepted for the Final Exam.** Do not email us with a request for late submission, it is not acceptable.
- No make-up exam will be offered.
- Individual submission is required from each student. Every student MUST work on their own submission.
- It is your responsibility to make sure your answer sheet is properly submitted via Canvas, before the deadline. **Double check** that you have submitted the correct answer sheet. No answer sheet will be accepted after the submission deadline.
- **Any sort of plagiarism/cheating will result in severe penalties**.

Questions

Q-1. Explain the following statement: “Semaphore can provide a more sophisticated way of process synchronization, as compared to Mutex Lock.”

Ans-1. Semaphore and mutex locks are different ways to control access to a shared resource when several processes or threads are concurrently executing. Semaphore has the advantage of multiple threads being able to access the critical section of code at the same time whereas with a mutex lock, only one thread at a time can access the shared resource. Because of this, semaphores tend to be more flexible in controlling access to multiple resources at the same time whereas mutex locks may not be able to fully utilize all resources available. Semaphores

essentially have more capabilities when it comes to synchronization, including preventing deadlock and different operations for more explicit signaling which means a semaphore can better implement synchronization patterns.

Q-2. Considering the security guarantee and performance overhead, state one prevention technique that you would choose (among the techniques discussed in lecture) for gaining protection against both direct and indirect buffer overflow attacks. Explain the rationale behind your choice.

Ans-2. Out of all the prevention techniques we have discussed in class, I would choose compiler modifications in the form of a Return Address Defender to protect against buffer overflow attacks. Though some performance overhead does incur from increased compile time and other performance overhead, I would consider it a worthwhile investment. Many prevention techniques are focused on direct buffer overflow attacks, this compiler technique covers both direct and indirect attacks. If the implementation was robust enough, I would choose this compiler modification technique based on the broad coverage of attacks against return addresses.

Q-3. Consider that there are three memory frames; according to LRU page replacement algorithm, how many page faults would occur for this reference string: 5,1,4,3,0,2,1,3,0,5,4,0,1,0,5,2,1,2,3, 1, 0, 5, 3? **Note**: You must show the steps how you calculated the page faults.

Ans-3. 19 page faults would occur for the reference string calculated like so:

Reference String: 5,1,4,3,0,2,1,3,0,5,4,0,1,0,5,2,1,2,3, 1, 0, 5, 3

5	**5**	**5**	**3**	**3**	**3**	**1**	**1**	**1**	**5**	**5**		**1**		**1**	**2**	**2**		**2**		**0**	**0**	**0**
	1	**1**	**1**	**0**	**0**	**0**	**3**	**3**	**3**	**4**		**4**		**5**	**5**	**5**		**3**		**3**	**5**	**5**
		4	**4**	**4**	**2**	**2**	**2**	**0**	**0**	**0**		**0**		**0**	**0**	**1**		**1**		**1**	**1**	**3**

Q-4. For passing parameters from a user program to the operating system during context switching, which method would you prefer most, and why?
Ans-4. For passing parameters, I prefer the stack method because, though it is slower than passing in registers, it is more flexible and easier to implement. It

seems to be a happy medium between the higher storage but slower memory access of the block method and the low storage but faster register access of the register method. Of course, a combination of the register passing and block method would be the best because you could pull the best qualities (the speed and memory) from both methods like the Linux operating system does. However, if you are specifically choosing from the three methods, I think I would choose the stack method.

Q-5. Describe how the ‘mailbox’ is used in indirect communication between processes.

Ans-5. In indirect communication, A process is not interrupted and messages are communicated through a “mailbox”. While a process is running, the mailbox resides in the address space of the process. As soon as the process terminates, the mailbox disappears along with deallocation of address space. It is like the operating system owns the mailbox and assigns it to a new process once one process has terminated.

Q-6. Explain the deadlock situation in the resource-allocation graph shown in Figure 1. Here, T1, T2, and T3 are threads – presented by circles; R1, R2, and R3 are resource types – presented by rectangles. Each resource type has multiple instances, where each instance is presented by a dot within the rectangle.

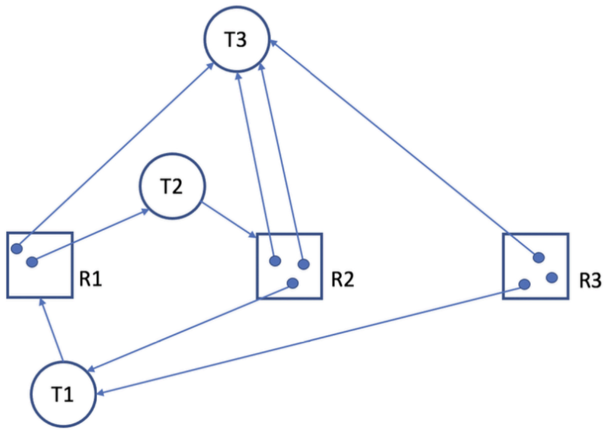

Figure 1: Resource-allocation Graph

Ans-6. In this deadlock situation, it seems like T3 and T1 are holding onto the R2 resources while T2 is waiting for the resource. However, T1 cannot process because it is waiting for the R1 resource which is currently being held by T3 and T2 so T1 cannot run because it is waiting on T2 and T2 cannot run because it is waiting on T1. Currently it seems like T3 is holding many of the resources to run which prevents T2 and T1 from running.

Q-7. Which method would you prefer most for dynamic storage allocation when a system has to satisfy a request of memory size n from a list of available holes? Why would you prefer this method?

Ans-7. When a system has to satisfy a request of memory size n from a list of available holes for dynamic storage allocation, I would prefer the first fit method. In terms of storage utilization, both first fit and best fit are better than worst fit. However, there is no substantial difference between first fit and best fit. Because there isn't really a storage difference, I would prefer the first fit because it is more straightforward to implement.

Q-8. Consider five processes with given burst time: P1 (burst time: 24 units of time), P2 (burst time: 3 units of time), P3 (burst time: 10 units of time), P4 (burst time: 5 units of time), and P5 (burst time: 3 units of time). Here, Time Quantum is 4 units of time. Initially, processes are in the ready queue in this order: P1 (first process in the ready queue), P2, P3, P4, P5 (last process in the ready queue). The time for context switch is 1 unit of time. Answer the following questions based on Round-Robin scheduling algorithm:

Ans-8.

a. **Draw a Gantt chart to demonstrate the order in which processes will be selected for scheduling.**

PROCESS	BURST TIME
P1	24, 20, 26, 12, 8, 4, 0
P2	3, 0
P3	10, 6, 2, 0
P4	5, 1, 0
P5	3, 0

P1	P2	P3	P4	P5	P1	P3	P4	P1	P3	P1	P1	P1

0 4 7 11 15 18 22 26 27 31 33 37 41 45

0 5 9 14 19 23 28 33 35 40 42 46 50 54

(second row accounts for context switch time)

b. **What will be the waiting time for each of five processes?**
c. **What will be the turnaround time for each of five processes?**

PROCESS	WAIT TIME	WAIT TIME (with context Switch)	TURNAROUND TIME
P1	23	27	5.75
P2	4	5	1
P3	23	30	5.75
P4	22	28	5.5
P5	15	19	3.75

d. **What will be the average turnaround time, if we add the time for context switch to the turnaround time?**

(27+5+30+28+19) / 4 = 27.25 time units

Q-9. In which cases, does a process switch from its waiting state to the ready state and is put back in the ready queue?

Ans-9. A process switches from its waiting state to the ready queue typically when a process is completed or a resource the process needs has become available. This could include an I/O operation completion event.

Q-10. Consider five threads: T0, T1, T2, T3, T4; and three resource types: A, B, C. Here, A has 12 instances, B has 8 instances, and C has 9 instances. The current state of the system is presented on Table 1, where 'Allocation' represents the number of instances for each resource type currently allocated to each thread, and 'Max' represents the maximum demand of each thread. Answer the following questions based on Banker's Algorithm:

Ans-10.

a. **Is the system in a safe state now?**
    i. **Yes, no more resources are allocated then what is available.**
b. **If the system is in a safe state, what is the corresponding safe sequence? It would suffice to present one safe sequence in your answer.**
    i. **Sequence: T1->T0->T3->T4->T2**

Threads	Allocation A, B, C	Max A, B, C
T0	3, 1, 1	10, 1, 2
T1	2, 1, 1	3, 4, 4
T2	1, 2, 1	6, 4, 2
T3	1, 0, 2	4, 2, 4
T4	1, 1, 1	3, 4, 3

Table 1

Q-11. In the context of memory allocation, which method would you use to tackle the external fragmentation issue? Why can/can't you use the same method to address internal fragmentation?

Ans-11. To deal with the external fragmentation issue, I would use the compaction method. Compaction consolidates free memory to create larger blocks of free memory. You can't use compaction to address internal fragmentation because it rearranges existing memory blocks which does not help with the issue of not having enough allocated space within the memory blocks. Because it does not focus on optimizing the use of individual blocks, you can't use it to address the internal fragmentation issue.