Name: ....
Student Number ....

COMPSCI 3SH3

DURATION OF EXAMINATION: 2.5 Hours — Dr. Bojan Nokovic
McMaster University Final Examination — April 22, 2021

1. (2 marks) Consider a logical address space of 1,024 pages with a 4-KB page size, mapped onto a physical memory of 512 frames.

   I) How many bits are required in the logical address?

   II) How many bits are required in the physical address?

   Answer:
   a.
   Since there are 1024 pages, this is 2^{10} pages, so 10 bits are
   required to represent the page indexing
   4KB page size can be represented with 2^{12}, so 12 bits are required for offset.
   We can use the formula:
   number of pages = address size / page size
   2^{10} = address size / 2^{12}
   and address size is 2^{22},
   so 22 bits are required for logical address

   b.
   512 can be represented as 2^9, so that means that 9 bits are required for indexing
   the frame number. We still need 12 bits for the offset,
   so in total we need 21 bits for the physical address.

2. (2 marks) Suppose that a disk drive has 5,000 cylinders, numbered 0 to 4,999. The drive is currently serving a request at cylinder 2,150, and the previous request was at cylinder 1,805. The queue of pending requests, in FIFO order, is: 2,069; 1,212; 2,296; 2,800; 524; 1,618; 256; 1,523; 4,965; 3,681 Starting from the current head position, what is the total distance (in cylinders) that the disk arm moves to satisfy all the pending requests for each of the following disk-scheduling algorithms?

   I) FCFS

   II) SCAN

   Answer:
   a. The FCFS schedule is 2,150; 2,069; 1,212; 2,296; 2,800; 524; 1,618; 356; 1,523; 4,965; 3,681 The total seek distance is (2150-2069)+ (2069-1212)+(2296-1212)+(2800-2296)+(2800-524)+(1618-524)+(1618-256)+(1523-256)+(4965-1523)+(4965-3681)=
   81+ 857 + 1084 + 504 + 2276 + 1094 + 1362 + 1267 + 3442 + 1284 =

938 + 1588 + 3370 + 2629 + 4726 = 13251

b. The SCAN schedule is 2,150; 2,296; 2,800; 3,681; 4,965; 2,069; 1,618; 1,523; 1,212; 524, 256.
The total seek distance is (4999-2150) + (4999-256) = 2849 + 4743 = 7592.

3. (2 marks) Consider a computer system with a 32-bit logical address and 4-KB page size. The system supports up to 2 GB of physical memory. How many entries are there in each of the following?

   I) A conventional, single-level page table

   II) An inverted page table

   Answer:
   a. $2^{32}/4\text{KB} = 2^{32}/2^{12} = 2^{20} = 1048576$ entries.
   b. $2\text{GB}/4\text{KB} = 2^{31}/2^{12} = 2^{19} = 524288$ entries.

4. (2 marks) Consider a file system that uses inodes to represent files. Disk blocks are 8 KB in size, and a pointer to a disk block requires 4 bytes. This file system has eight direct disk blocks, as well as single and triple indirect disk blocks.

   I) What is the maximum size of a file that can be stored in this file system?

   II) What is the maximum size of a file that can be stored in this file system if disk blocks are 4 KB and and a pointer to a disk block requires 2 bytes?

   Answer:

   I) (8 * 8KB) + (8KB/4bytes * 8KB) + (8KB/4bytes * 8KB/4bytes * 8KB/4bytes * 8KB) =
   (8 * 8KB) + (2048 * 8KB) + (2048 * 2048 * 2048 * 8KB)

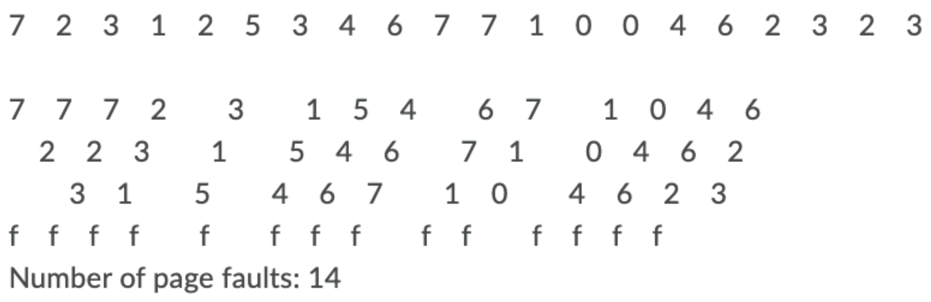
   II) (8 * 4KB) + (4KB/2bytes * 4KB) + (4KB/2bytes * 4KB/2bytes * 4KB/2bytes * 4KB) =
   (8 * 4KB) + (2048 * 4KB) + (2048 * 2048 * 2048 * 4KB)

5. (2 marks) Consider the following page reference string: 7, 2, 3, 1, 2, 5, 3, 4, 6, 7, 7, 1, 0, 0, 4, 6, 2, 3, 2,3. Assuming demand paging with three frames, how many page faults would occur for the following replacement algorithms?

   I) FIFO replacement

   II) LRU replacement

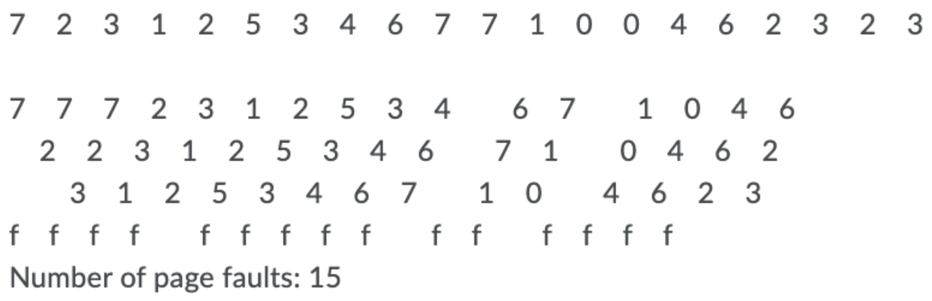
   a) FIFO



```
7  2  3  1  2  5  3  4  6  7  7  1  0  0  4  6  2  3  2  3

7  7  7  2     3     1  5  4     6  7     1  0  4  6
   2  2  3     1     5  4  6     7  1     0  4  6  2
      3  1     5     4  6  7     1  0     4  6  2  3
f  f  f  f     f     f  f  f     f  f     f  f  f  f
Number of page faults: 14
```

b) LRU



```
7  2  3  1  2  5  3  4  6  7  7  1  0  0  4  6  2  3  2  3

7  7  7  2  3  1  2  5  3  4     6  7     1  0  4  6
   2  2  3  1  2  5  3  4  6     7  1     0  4  6  2
      3  1  2  5  3  4  6  7     1  0     4  6  2  3
f  f  f  f     f  f  f  f  f     f  f     f  f  f  f
Number of page faults: 15
```

6. (2 marks)

Assume that we have a demand-paged memory. The page table is held in registers. It takes 8 milliseconds to service a page fault if an empty frame is available or if the replaced page is not modified and 20 milliseconds if the replaced page is modified. Memory-access time is 100 nanoseconds. Assume that the page to be replaced is modified 30 percent of the time.

I) What is the maximum acceptable page-fault rate if effective access time is no more than 200 nanoseconds?

II) What is the maximum acceptable page-fault rate if page to be replaced is modified 49 percent of the time and effective access time is no more than 1 microsecond?

Answer:

I) 0.2 microsec = (1 - P) 0.1 microsec + (0.7P) 8 millisec + (0.3P) 20 millisec
0.1 =0.1Pmicroseconds + 5600 P + 6000 P
0.1 = 11,600 P
P = 0.0000086

II) 1 microsecond = 1000 nanoseconds
1 microsecond = (1 - P) 0.1 microsec + (0.51P) 8 millisec + (0.49P) 20 millisec
1 microsecond = 0.1 - 0.1P + 4080P + 9800P
0.9 = 13380P
P = 0.0000672

7. **Threads** (4 marks)

a) (2 marks) What is the output of the program shown in Listing 1?

Listing 1: Processes Synchronization

```c
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>
#include <assert.h>

const size_t NUMTHREADS = 2;
int done = 0;
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t cond = PTHREAD_COND_INITIALIZER;

void* ThreadEntry(void* id)
{
  const int myid = (long)id;
  printf("Thread %d  working \n", myid);
  sleep(1);
  pthread_mutex_lock(&mutex);
  done++;
  printf("Thread %d done! \n", myid);
  pthread_cond_signal(&cond);
  pthread_mutex_unlock(&mutex);

  return NULL;
}

int main(int argc, char** argv)
{
  printf("Main thread starting\n");

  pthread_t threads[NUMTHREADS];

  pthread_create(&threads[0], NULL, ThreadEntry, (void*)(long)0);
  pthread_create(&threads[1], NULL, ThreadEntry, (void*)(long)1);

  pthread_mutex_lock(&mutex);

  while( done < NUMTHREADS )
  {
      printf("Main thread is waiting on cond.\n");
      pthread_cond_wait(&cond, &mutex);
      printf("Main thread  cond was signalled.\n");
  }

  printf("Main thread is done\n");
  pthread_mutex_unlock(&mutex);

  return 0;
}
```

Answer: One of the many possible outputs from running this program:
Main thread starting
Main thread is waiting on cond
Thread 0 working
Thread 1 working
Thread 0 done!
Thread 1 done!
Main thread cond was signalled.

Main thread is done

Comment: *The output depends on the other of execution. At the beginning, there will always be a "Main thread starting" There will always be some order of the strings "Thread 1 working" and "Thread 0 working" and ,"Thread 1 done", and "Thread 0 done", where the "working" of the thread must come before the "done", At the end, there must be a "Main thread is done". However the main thread will not finish before thread 0 and thread 1 are done, because of the while loop.*

b) (1 mark) Is deadlock possible at this program? Explain.
Answer:
No
Explanation: main() acquires the lock, then immediately checks the condition (which hasn't been signaled yet). Since it hasn't been signaled, `pthread_cond_wait()` releases the mutex and blocks the main() thread until the condition variable is signaled by another thread.

c) (1 mark) Is it possible that "Thread 1" completes its job before "Thread 0"? Why yes or why not?
Answer: Yes
Even though we have started "Thread 0" before "Thread 1" the thread scheduler may not start and end them in the specified order.
Possible output:
Main thread starting
Thread 0 working
Thread 1 working
Main thread is waiting on cond.
Thread 1 done!
Thread 0 done!
Main thread cond was signalled.
Main thread is done

8. **Processes** (4 marks) Write C code that produces process tree shown in Figure 1.

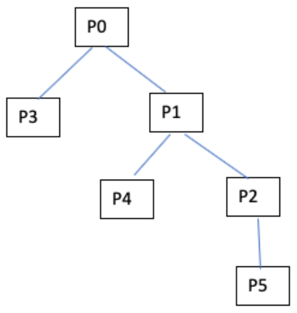


Figure 1: Process Tree

Answer:

Listing 2: Fork Example

```
main(){
  pid_t pid;
  pid = fork();
  if (pid == 0) {
    /* child process */
      fork();
  }
  fork();
}
```

9. **Race Conditions** (8 marks)

Listing 3 shows implementation of infinite buffer Producer-Consumer using binary semaphores.

Listing 3: Infinite-Buffer Producer-Consumer

```
int n = 0;
binary_semaphore  s = 1, d = 0;

void procucer(){
  while (true){
    // ... produce item
    Wait(s);
    //... append it to buffer
    n++;
    if(n==1)
      Signal(d);
    Signal(s);
  }
}

void consumer(){
  Wait(d);
  while (true){
    Wait(s);
    // ... take item from buffer
    n--;
    Signal(s);
    // ... consume one item
    if(n==0)
      Wait(d);
  }
}

void main() {
   n=0;

   parbegin(producer, consumer);
}
```

(a) (4 marks) For lines 2 - 21 of Table 1 fill in values of the last three columns, as they would appear after instructions shown are executed.

Answer:

Table 1: Possible Scenario for Infinite-Buffer Producer-Consumer Program

Instruction #	Producer	Consumer	s	d	n
1			1	0	0
2	Wait(s)				
3	n++				
4	if(n==1) Signal(d);				
5	Signal(s)				
6		Wait(d)			
7		Wait(s)			
8		n - -			
9		Signal(s)			
10	Wait(s)				
11	n++				
12	if (n==1) Signal(d)				
13	Signal(s)				
14		if(n==0) Wait(d)			
15		Wait(s)			
16		n - -			
17		Signal(s)			
18		if(n==0) Wait(d)			
19		Wait(s)			
20		n - -			
21		Signal(s)			

Instruction #	Producer	Consumer	s	d	n
1			1	0	0
2	Wait(s)		0	0	0
3	n++		0	0	1
4	if(n==1) Signal(d);		0	1	1
5	Signal(s)		1	1	1
6		Wait(d)	1	0	1
7		Wait(s)	0	0	1
8		n - -	0	0	0
9		Signal(s)	1	0	0
10	Wait(s)		0	0	0
11	n++		0	0	1
12	if (n==1) Signal(d)		0	1	1
13	Signal(s)		1	1	1
14		if(n==0) Wait(d)	1	1	1
15		Wait(s)	0	1	1
16		n - -	0	1	0
17		Signal(s)	1	1	0
18		if(n==0) Wait(d)	1	0	0
19		Wait(s)	0	0	0
20		n - -	0	0	-1
21		Signal(s)	1	0	-1

(b) (3 marks) Indicate critical sections (by instructions numbers) controlled by semaphore s. Answer: 3, 4, 8, 11, 12, 16, 20.

(c) (1 marks) What integer n represents? Can n become negative? If yes what negative n means? Answer: Yes, negative n means the consumer has consumed the item that does not exists, a fatal flow.

10. **Deadlocks** (4 marks)
Consider the following snapshot of a system:

Process	*Allocation*	*Max*
	ABCD	ABCD
T_0	1 2 0 2	4 3 1 6
T_1	0 1 1 2	2 4 2 4
T_2	1 2 4 0	3 6 5 1
T_3	1 2 0 1	2 6 2 3
T_4	1 0 0 1	3 1 1 2

Using the banker's algorithm, determine whether or not each of the following states is unsafe. If the state is safe, illustrate the order in which the threads may complete. Otherwise, illustrate why the state is unsafe.

a) Available = (3,0,1,4)

b) Available = (2,2,2,3)

8.27 Answer:

a) No. It is unsafe since all threads need some amount of resource B, but there is none in the starting state, and no thread will release their resources by themselves.

b) Yes. One possible ordering is T4, T0, T1, T2, T3
Total available resources After T4 : 3 2 2 4
Total available resources After T0 : 4 4 2 6
Total available resources After T1: 4 5 3 8
Total available resources After T3: 5 7 3 9
Total available resources After T2: 6 9 7 9
All threads completed

11. **CPU Scheduling** (8 marks)

The following processes are being scheduled using a preemptive, round-robin scheduling algorithm.

Process	*Priority*	*Burst Time*	*Arrival*
P_1	40	20	0
P_2	30	25	25
P_3	30	25	30
P_4	35	15	60
P_5	5	15	95
P_6	10	15	100

Each process is assigned a numerical priority,with a higher number indicating a higher relative priority. In addition to the processes listed below, the system also has an idle task (which consumes no CPU resources and is identified as P_{idle}). This task has priority 0 and is scheduled when-ever the system has no other available processes to run. The length of a time quantum is 10 units. If a process is preempted by a higher-priority process, the preempted process is placed at the end of the queue.

(a) (2 marks) Show the scheduling order of the processes using a Gantt chart. To represent Gantt chart use notation "[*range*] → *process*". For example the chart shown in Figure 2

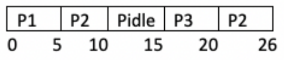


Figure 2: Gantt Chart Exampe

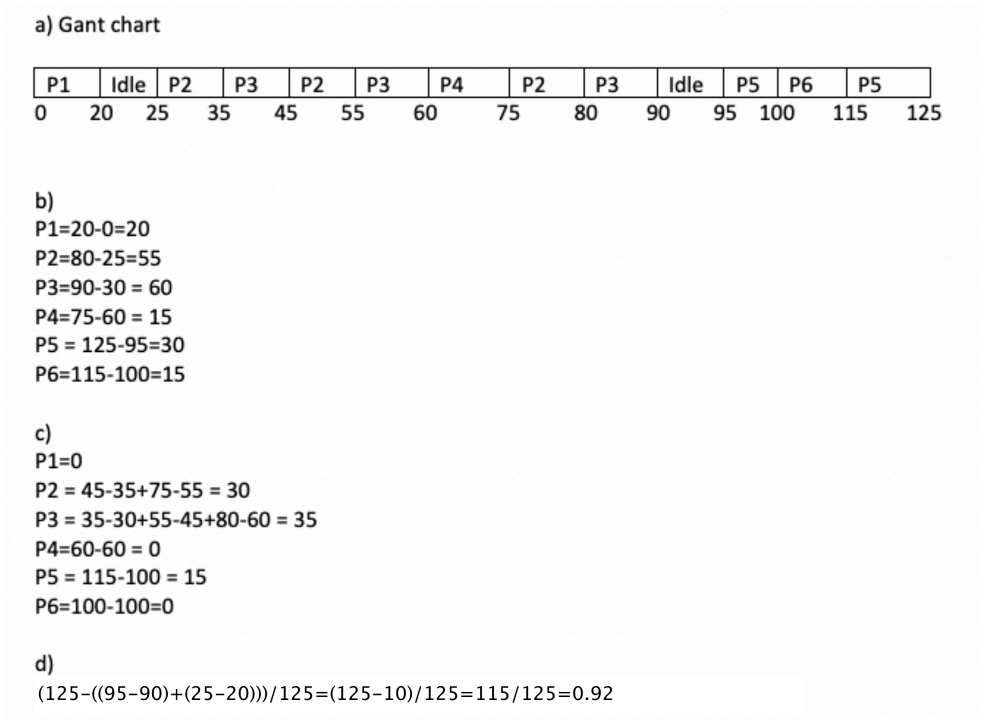
is represented as: [0-5] → P1, [5-10] → P2, [10-15] →Pidle, [15-20] → P3, [20-26] → P2.

(b) (2 marks) What is the turnaround time for each process?

(c) (2 marks) What is the waiting time for each process?

(d) (2 marks) What is the CPU utilization rate?

Answer:



a) Gant chart

P1
Idle
P2
P3
P2
P3
P4
P2
P3
Idle
P5
P6
P5
0
20
25
35
45
55
60
75
80
90
95
100
115
125

b)
P1=20-0=20
P2=80-25=55
P3=90-30 = 60
P4=75-60 = 15
P5 = 125-95=30
P6=115-100=15

c)
P1=0
P2 = 45-35+75-55 = 30
P3 = 35-30+55-45+80-60 = 35
P4=60-60 = 0
P5 = 115-100 = 15
P6=100-100=0

d)
(125–((95–90)+(25–20)))/125=(125–10)/125=115/125=0.92

Figure 3

The End