ASSIGNMENT 1 CMPT 300 SPRING 2019

1. **[36 points]** You will write a C program that will require you to practice creating and managing multiple generations of processes. In the description of the program below words in all capital letters should be replaced with the actual values of the process IDs when the line is printed. Use rand() and srand() from stdlib.h to generate random numbers needed below. Please use the printed outputs given below without editing, each of the printed outputs should begin on a new line when printed as output. Be sure to deal with potential errors.

   The parent process (the first process created when the program is started) will execute the following steps

   a) Prompt "enter the seed for the parent process ". Then read the seed. The seed will be used to determining numbers of children and waiting times.
   b) Initialize the random number generator using srand(). Then use rand to determine the number of children the process will generate (5 <= number of process <=9)
   c) Determine the process id of the running process.
   d) Print "My process ID is PARENTPID"
   e) Before each child is generated the parent process will print "MYPROCESSID is about to create a child"
   f) Create the child
   g) After each child process is generated the parent process will print "Parent MYPROCESSID has created a child with process ID CHILDPROCESSID"
   h) After all children have been created consider each child in the same order the children were created.
      i. Print "I am the parent, I am waiting for child CHILDPROCESSID to terminate"
      ii. After the child CHILDPROCESSID has terminated print "I am process MYPROCESSID. My child CHILDPROCESSID" is dead"
   i) After the last child has died the process will print "I am the parent, child CHILDPROCESSID has terminated"
   j) Sleep for 5 seconds
   k) Terminate the process

   Any child created by the child process will be referred to as a grandchild.
   Each child process will

   a) Generate a seed for the child process by adding to the seed for the parent process. Add 0 for the first process generated, 1 for the second process generated, 2 for the third process generated and so on.
   b) Reinitialize the random number generator using the child seed and srand().
   c) Print "I am a new child, my process ID is CHILDPROCESSID, my seed id CHILDSSEED"
   d) Generate a random integer number 1 <= number <= 3. This random number will determine how many children this child process will generate.
   e) Prints "I am child CHILDPROCESSID, I will have NUMCHILDREN children"
   f) Before each grandchild is generated by the child process print "I am child CHILDPROCESSID, I am about to create a child"
   g) The child process creates a grandchild.

h) After each grandchild is generated the child process will print "I am child CHILDPROCESSID, I just created a child"

i) After all grandchildren have been created print
"I am the child CHILDPROCESSID, I have NUMGRANDCHILDREN children, "
"I am waiting for my children to terminate"

j) After all the child's children have been created consider each child in the same order the children were created. After each child GRANDCHILDPROCESSID has terminated print "I am child CHILDPROCESSID. My child GRANDCHILDPROCESSID has been waited". Wait for the first process, when the first process has terminated wait for the second and so on.

k) After all of the children have been waited print "I am child CHILDPROCESSID, I am about to terminate"

l) Sleep for 5 seconds

m) Terminate the process

Each grandchild process will

a) Print "I am grandchild GRANDCHILDPROCESSID, My grandparent is PROCESSID, My parent is PARENTPROCESSID"

b) Generate a random integer number 5 <= sleep number <= 14.

c) Make the grandchild sleep for sleep number seconds

d) After the grandchild wakes up print "I am grandchild GRANDCHILDPROCESSID with parent CHILDPROCESSID, I am about to terminate"

e) Terminate the process

2. Consider a system that uses a 64-bit word. This CPU has a 1 Mbyte (1024*1024 bytes) cache memory. The size of each CPU cache slot is 512 bytes. The system has a 4 GB main memory. Cache line 0 holds main RAM memory (byte) addresses 0 to 511, cache line 1 holds main memory (byte) addresses 512 to 1024, and so on. Assume that the cache initially contains the information in the first 550 cache lines of main RAM memory. Also assume that cache line 0 was loaded first, and that the other 549 cache lines were loaded in numerical sequence. After being loaded each of the cache lines was accessed several times, but the last access to cache line N occurred before the first access to cache line N+1. Further assume that the remainder of the cache (cache slots 550 …) are empty. In other words they have not been used to hold cache lines since the OS was started.

   Consider a program that makes a series of memory requests for data and/or instructions. Each request is for information stored in a particular cache line of main RAM memory. The part of the code we are considering consists of two consecutive loops. The first loop will be executed 8 times. The second loop is executed 43 times. To provide the needed instructions and data to the program, as it runs, cache lines are accessed in the following order:

   i. The code and data in the first loop is stored in cache lines numbered 400 to 950. Each time the first loop is executed there will be 5 accesses for each cache line, 5 accesses to cache line 400 then 5 accesses to cache line 401, and so on, …, finishing with 5 accesses to cache line 950.

   ii. The code and data in the second loop is stored in cache lines 4666 to 5028, and lines 16968 – 17314. Each time the second loop is executed there will be 3 accesses to cache line 4666, 3 accesses to cache line 4667, …, finishing the first group of cache lines with 3 accesses to cache line

5028, then there will be 3 access to cache line 16968, then 3 accesses to 16969, and so on until the 3 accesses to line 17314 are complete.

You will be asked to consider two different mapping algorithms. For each of these mapping algorithms assume the replacement algorithm is to place the new cache line in the MAPPED cache slot that was accessed least recently. The rules for the replacement algorithm are:

i. If there are MAPPED slots that are empty, choose the empty MAPPED slot with the smallest number

ii. otherwise choose the MAPPED cache slot that was last accessed the longest time ago (last accessed at the earliest time)

a) **[16 points]** The system uses direct mapping of the 4GB memory of the system to the cache. Direct mapping means that if there are N cache slots in the cache memory and M cache lines in memory, cache line K in memory will map to cache slot K%N in the cache. When cache line K is loaded in the cache it must always be loaded in cache slot K%N. What are the number of cache hits, the number of cache misses and the hit ratio for the series of accesses above using direct mapping? Give a step by step explanation in your answer which includes a summary of which cache lines are placed in which cache slots in which sequence. Most of the points will be given for your explanation.
HINT: *The cache initially contains the information in the first 550 cache lines of main memory in its first 550 cache slots (Cache line 0 in cache slot 0, cache line 1 in cache slot 1, …, finishing with cache line 549 in cache slot 549).*
HINT: *First consider how many hits and misses for the first time through the first loop, then consider how many hits and misses for the remaining times through the first loop, then consider the first time through the second loop, then the subsequent times through the second loop*

b) **[16 points]** Assume that all cache lines in the cache when the code begins to execute (lines 0 to 549) were loaded into the cache in numerical order using 4 way associative mapping. When we use M way set associative mapping (for this example M=4) we divide the available cache slots into sets of M slots. If there are N cache slots in the cache memory we can divide them into P groups of M slots (P*M=N). Cache line K in memory will map to any cache slot in group Q (0<=Q<P) if K%P = Q. When the cache is empty cache line 0 in stored cache slot 0 (first slot of group 0), cache line 1 is stored in cache slot 4 (first slot of group 1), cache line 2 is stored in cache slot 8 (first slot of group 2), and so on. Then cache line 512 is stored in cache slot 1 (second slot of group 0), cache line 513 is stored in cache slot 5 (second slot of group 1) and so on. When all slots in a group are full, then the slot in the group that was last accessed the longest time ago will be replaced. What is the hit ratio for the series of accesses above using 4 way set associative mapping of the cache? Give a step by step explanation in your answer which includes a summary of which cache lines are placed in which cache slots at what times

3. Three CPU bound jobs A through C and one I/O bound job D arrive at a computer center at the times shown in the table below. All times in the table are in ms. Ignore context switches. The system follows the following rules:
   a. In the priority column higher numbers indicate higher priorities
   b. The I/0 bound job runs for 3ms of CPU time then prints a value, this pair of actions repeat until the job finishes.

c. The print hardware takes 2ms to print one value.
d. The quanta for an I/O bound jobs is cumulative. Therefore:
   i. When an I/O bound job leaves the CPU the remaining time in its quanta is recorded
   ii. When an I/O bound job re-enters the CPU after completing a write the length of the quanta given to it is the remaining time in the quanta recorded before the job left the CPU to enter the print queue
e. When a job leaves the print queue it is placed in the ready queue
   i. If the job is starting a new quanta it is placed at the end of the ready queue
   ii. If it is continuing an unfinished quanta it is placed at the front of the ready queue
f. When a pre-empted job leaves the CPU it is placed at the front of the ready queue
g. If two actions take place at the "same" time the order they are completed is
   i. place a newly arrived process in the queue
   ii. check the queues to see which process is loaded into the CPU next
   iii. place a process leaving the CPU into the ready queue or the print queue
   iv. place a process leaving the print queue into the ready queue
h. Each process is allowed a quantum of 6 ms of CPU. At the end of the quantum the process will be placed at the end of the appropriate ready queue
   i. For CPU bound jobs this quantum will be used in one visit to the CPU.
   ii. For I/O bound jobs this quantum may be distributed over several visits to the CPU.

Process	Arrival time (ms)	Running Time (ms)	Priority
A	12	28	4
B	19	34	7
C	2	36	4
D	7	9	7

Determine the turnaround time for each process and the average turnaround time for all processes for each of the following scheduling algorithms. Explain how you arrived at your answers. In particular, your explanation should include a COMPLETE list of steps showing the order in which the processes execute. The duration of each step, and the elapsed time at the end of each step should be shown. The description may be presented as a bulleted list of steps OR a table, with one column per process and an entry for each step.

a) **[13 points]** round-robin scheduling (no priorities)
b) **[13 points]** round-robin scheduling (with preemptive priorities). ***When a job arrives in the high priority queue, the running job is checked to see if it has a lower priority. If the running job has a lower priority it will be immediately replaced with the higher priority job***
c) **[6 points]** first come first served (with non-preemptive priorities).
   NOTE: NO TIME SHARING (not round robin)