Operating System (CSE231)

Section-B

Mid-semester Examination - **RUBRIC**

Total Marks: 100 **Time: 1.5 hrs**

---

1. (8 points) For the following C program, named `sample.c`, determine the various process states and when they occur (mention code line number).

```
0   #include <stdio.h>
1   #include <unistd.h>
2
3   int main() {
4       printf("Start\n");
5       sleep(3);
6       printf("Continuing\n");
7       return 0;
8   }
```

Assume the executable is `sample` and the following line has been put out on the terminal.

`./sample`

Along with the process states, indicate briefly the steps that the OS takes during process creation and termination (where appropriate).

Line No.	**Code**	**Process State**	**OS Action/Notes**
–	./sample	**Ready**	Create a process and allocate memory for it
3	`int main(){`	**Ready -> Running**	–
4	`printf("Start");`	**Running**	–
5	`sleep(3);`	**Running -> Blocked**	Waiting on timer
–	*(timer expires)*	**Blocked -> Ready**	Moves to ready queue
6	`printf("Continuing");`	**Ready -> Running**	Scheduled again, executes
7	`return 0;`	**Running**	–
8	}	**Terminated**	Process terminates; OS deallocates the memory for this process and cleanup

Grading guide: 1 full mark per line of the table with correct line number, process state, and OS Action.

2. (10 points) Complete the following C program so that it creates a child process which redirects standard input from the file ‘‘numbers.txt" and executes the system ‘‘sort" command on the numbers using `execvp`. The parent waits for the child to finish. Fill in the blanks with the correct system calls, parameters, and explain them briefly. Note that standard file descriptors in C are 0 for stdin, 1 for stdout, and 2 for stderr.

```c
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>
#include <fcntl.h>
#include <stdlib.h>

int main() {
    int rc = fork();
    if (rc < 0) {
       fprintf(_______, "fork failed\n");
        exit(1);
     } else if (rc == 0) {
     // Child: redirect standard input to a file
       close( ______ ); //
       open(_______, O_RDONLY);
       char *args[] = {"sort", NULL};
       execvp("sort", ______ );
       perror("exec failed");
     } else {
       // Parent process
       _______ ;
       printf("Sorting completed\n");
     }
     return 0;}
```

(a) `stderr`
Error messages are normally written to standard error.
(b) `STDIN_FILENO` (or 0)
Closes the standard input file descriptor, frees up fd 0 (stdin).
(c) `"numbers.txt"`
Opens the file in read-only mode. Since stdin (fd 0) was closed, open() assigns fd 0 to this file.
(d) `args`
Passes the argument vector to execvp, which replaces the child image with the sort image, which now reads from stdin (the file numbers.txt).
(e) `wait(NULL);`
Parent process waits for the child to terminate.

Grading guide: 2 full marks for correct answer and explanation. Only one mark if no explanation is given.

3. (16 points) Given a system with a virtual address space of 8 bits, where the page size is 64 bytes, the following page table for a process is known (where -1 is an invalid entry):

VPN	PFN
0	2
1	-1
2	1
3	0

Translate the following virtual addresses (decimal) into physical addresses, or indicate TRAP if invalid:

1. 45
2. 90
3. 150
4. 200

1. 45 = 00 101101
Append PFN to Offset and convert to Decimal
PFN = 2 = $(10)_2$
Therefore PA = 10101101 = $(173)_{10}$
Students may also use the following method for calculating PA:
VPN = 00, which is PFN 2 from the page table, and offset = 101101 = 45, then PA = $2 \times 64 + 45 = 173$.

2. 90 = 01 011010
VPN = 1 = 1
Offset = 011010 = 26
PFN lookup: VPN 1 = PFN = -1
TRAP.

3. $(150)_{10} = (10010110)_2$
VPN = 10 = 2
Offset = 010110 = 22
PFN Lookup = VPN 2 = PFN 1
PFN = 1 = 01
Therefore PA = 01010110 = 86

4. $(200)_{10} = (11001000)_2$
VPN = 11 = 3
Offset = 001000 = 8
PFN lookup = VPN 3 = PFN 0
PFN = 0 = 00
Therefore PA = 00001000 = 8

Note to TAs: The students may consider the VA space to be 8-bits or 8 bytes, you may consider both for the answer in the following way:

1. VA space = 8 bits; Page size = 64 bytes
2. VA space = 64 bits (8 bytes); Page size = 64 bytes

In the first case, of the 8 bits, the lower 6 bits are offset, and the upper two bits are the VPN. This VPN is mapped to a PFN using the table, converted back to binary, and appended to the existing offset to give the PA (as calculated above).

In the second case, if the VA space is 8 bytes, that would mean, of the 64 bits, the lower 6 bits would be offset, and the remaining 58 bits would be the VPN. This seems to be overkill, as the binary conversions of the asked questions (45, 90, 150, and 200) are (101101, 1011010, 10010110, and 11001000) respectively (all less than/equal to 8 bits in length). Doing the calculation would mean prefixing at least 54 0's to these binary numbers to make the same calculations. *Regardless*, the answers should be the same, as we ignore extra zeroes, and the VPN does not change. The final answers should be correct.

Another case is if the student has considered the page size to be 64 bits, then it should generate a TRAP in all cases. This is because the minimum VPN in such a consideration is 5, and since the values given in the page table go up to max 3, every case would lead to a trap due to no appropriate VPN to PFN mapping. But in order to get marks, students must provide this reasoning.

Grading guide: 4 full marks for each part. 1 mark for decimal to binary conversion of the VA. 1 mark for correct mapping/lookup from the table. 2 marks for correct calculation of the PA (either using the appending method or the formula method).

4. (24 points) The size of main memory is 1000 KB. Initially the free holes (address ranges and sizes) are as follows.

- H1: 0–99 (100 KB)
- H2: 100–299 (200 KB)
- H3: 300–349 (50 KB)
- H4: 350–599 (250 KB)
- H5: 600–699 (100 KB)
- H6: 700–999 (300 KB)

Processes arrive and must be allocated segments using first-fit (scan holes from low address to high for each segment and allocate the first available free hole that fits the segment). Allocate the following processes in order (segments listed in the order they must be placed).

- P1: code = 120 KB, data = 80 KB, stack = 40 KB
- P2: code = 200 KB, data = 150 KB
- P3: code = 130 KB, data = 70 KB, heap = 20 KB
- After the above allocations, a new process P4 arrives with a single segment: S1 = 190 KB.

Answer the following.

(a) (16 points) Show the allocations (addresses) and the resulting memory map after P1, P2, P3 are placed.

We use first-fit for every segment (scan holes from lowest address). Initial holes are as follows.
H1: 0–99 (100)
H2: 100–299 (200)
H3: 300–349 (50)
H4: 350–599 (250)
H5: 600–699 (100)
H6: 700–999 (300)

1. Allocate P1 (code 120, data 80, stack 40)
   P1.code (120 KB): H1 (100) too small → H2 (200) fits. Place P1.code at addresses 100–219 (120 KB). Remaining of H2 is 220–299 (80 KB).

   P1.data (80 KB): Start again at H1: H1 (100) fits. Place P1.data at 0–79 (80 KB). Remaining of H1 is 80–99 (20 KB).

   P1.stack (40 KB): Scan: H1 rem (20) too small → H2 rem (80) fits. Place P1.stack at 220–259 (40 KB). Remaining of H2 becomes 260–299 (40 KB).

   After P1 allocations, allocated blocks are as follows.
   - P1.data: 0–79
   - (H1 rem) 80–99 (20 free)
   - P1.code: 100–219
   - P1.stack: 220–259
   - (H2 rem) 260–299 (40 free)
   - H3: 300–349 (50)
   - H4: 350–599 (250)
   - H5: 600–699 (100)
   - H6: 700–999 (300)
2. Allocate P2 (code 200, data 150).
   P2.code (200 KB):
   H1 remaining 20 (no); H2 remaining 40 (no); H3 50 (no); H4 250 fits. Place P2.code at 350–549 (200 KB). Remaining of H4 becomes 550–599 (50 KB).

   P2.data (150 KB):
   Scan: H1 20 no; H2 40 no; H3 50 no; H4 rem 50 no; H5 100 no; H6 300 fits. Place P2.data at 700–849 (150 KB). Remaining of H6 becomes 850–999 (150 KB).
3. Allocate P3 (code 130, data 70, heap 20).
   P3.code (130 KB):
   Scan holes: H1 20 no; H2 40 no; H3 50 no; H4 rem 50 no; H5 100 no; H6 rem 150 fits. Place P3.code at 850–979 (130 KB). Remaining of H6 becomes 980–999 (20 KB).

   P3.data (70 KB):
   Scan: H1 20 no; H2 40 no; H3 50 no; H4 rem 50 no; H5 100 fits. Place P3.data at 600–669 (70 KB). Remaining of H5 becomes 670–699 (30 KB).

P3.heap (20 KB):
Scan: H1 has 20 KB (80–99) – fits exactly. Place P3.heap at 80–99 (20 KB). H1 now fully used.

Memory map after P1, P2, P3 allocations. Allocated blocks (in increasing address) are as follows.

- 0–79 : P1.data (80)
- 80–99 : P3.heap (20)
- 100–219 : P1.code (120)
- 220–259 : P1.stack (40)
- 260–299 : free (40) — leftover of H2
- 300–349 : free (50) — H3
- 350–549 : P2.code (200)
- 550–599 : free (50) — leftover of H4
- 600–669 : P3.data (70)
- 670–699 : free (30) — leftover of H5
- 700–849 : P2.data (150)
- 850–979 : P3.code (130)
- 980–999 : free (20) — leftover of H6

Grading guide: P1 has 3 segments, P2 has 2 segments, and P3 has 3 segments. Allot one point each for the correct allocation (8 points). The rest of the 8 points for the correct memory map.

(b) (4 points) Can P4.S1 be allocated with first-fit? Explain.

No. First-fit requires a single contiguous hole of at least 190 KB for this single segment. The largest contiguous free block is only 50 KB. Although the total free memory = 190 KB, it is split into several small holes. Because segmentation requires contiguous placement per segment, there is no single hole $\geq$ 190 KB. Therefore the allocation fails.

Grading guide: 1 point for the right answer and 3 points for the right explanation.

(c) (4 points) Compute the free memory holes after the allocation and also the total free memory. What is this scenario (P4.S1 memory request, free memory holes, total free memory) referred to as? Explain.

The current free holes (contiguous free blocks) are as follows.

- F1: 260–299 = 40 KB
- F2: 300–349 = 50 KB
- F3: 550–599 = 50 KB
- F4: 670–699 = 30 KB
- F5: 980–999 = 20 KB

Total free = 40 + 50 + 50 + 30 + 20 = 190 KB

This scenario is called external fragmentation. If we add up all the small holes, it is sufficient to address the memory request from P4.S1. But, individually, none of the small holes are big enough to accommodate the memory request. This is called external fragmentation.

Grading guide: 1 point for listing the free memory holes correctly (0.2 mark for each). 1 point for getting the total free memory correct. 1 point for recognizing that this is external fragmentation and 1 point for the correct explanation.

5. (20 points) A system has a TLB that can hold a maximum of 4 entries. The TLB uses the Least Recently Used (LRU) replacement policy. Initially the TLB is empty. A process generates the following sequence of virtual page numbers (VPNs):

   1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

   The TLB lookup time is 1 ns. Main memory access time is 50 ns. The time to add an entry to TLB can be ignored. Assume that the page table resides in main memory. Answer the following.

   (a) (12 points) For each VPN in the sequence, state whether it is a TLB hit or TLB miss. Show the contents of the TLB after each reference (show only the VPN and ignore other aspects). List the entries from most recently used to least recently used.

Step	VPN	Hit/Miss	TLB Contents
1	1	Miss	[1]
2	2	Miss	[2, 1]
3	3	Miss	[3, 2, 1]
4	4	Miss	[4, 3, 2, 1]
5	1	Hit	[1, 4, 3, 2]
6	2	Hit	[2, 1, 4, 3]
7	5	Miss	[5, 2, 1, 4] (evict 3)
8	1	Hit	[1, 5, 2, 4]
9	2	Hit	[2, 1, 5, 4]
10	3	Miss	[3, 2, 1, 5] (evict 4)
11	4	Miss	[4, 3, 2, 1] (evict 5)
12	5	Miss	[5, 4, 3, 2] (evict 1)

Table 1: TLB contents using LRU policy

Grading guide: 1 point for each correct entry in the table (there are 12 entries which will result in 12 points max).

   (b) (4 points) Compute the total number of TLB hits and misses for the sequence.

Total references = 12
Total hits = 4
Total misses = 8

Grading guide: 2 points for getting the total hits correct and 2 points for getting the total misses correct. No partial grading here.

(c) (4 points) Using the hits and misses, compute the total memory access time and the average memory access time. Report both these values in nanoseconds. Show your calculation.

Hit cost = TLB lookup time + 1*memory time = 1 + 1*50 = 51 ns
Miss cost = TLB lookup time + 2*memory time + TLB lookup time (for the retry instruction) = 1 + 2*50 + 1 = 102 ns
Total memory access time = number of hits * hit cost + number of misses * miss cost = 4 * 51 + 8 * 102 = 1020 ns
Average memory access time = 1020/12 = 85 ns

Grading guide: 1 point for hit cost computation, 1 point for miss cost computation, 1 point for total memory access time, and 1 point for average memory access time.

6. (22 points) Lottery scheduling policy is used on four processes arriving at different times with the properties provided in Table 2.

Process	Arrival Time	Burst Time	Tickets
P1	0	3	2
P2	1	4	3
P3	2	2	1
P4	3	1	2

Table 2: Four jobs with their arrival time, run time and number of tickets.

One ticket is drawn per time unit. The process holding the winning ticket runs for 1 unit. Random winning draws are P1, P2, P1, P3, P2, P2, P4, P2, P1, P3. This starts from t = 0. If the chosen process has not arrived yet, redraw (ignore that draw). Compute the following (with steps).

(a) (12 points) Completion time of each process

Expand draws with arrival times and burst time (BT).

- t = 0: Only P1 has arrived → Draw = P1 → run P1 (BT left 2).
- t = 1: P1 and P2 available → Draw = P2 → run P2 (BT left 3).
- t = 2: P1, P2, P3 available → Draw = P1 → run P1 (BT left 1).
- t = 3: P1, P2, P3, P4 available → Draw = P3 → run P3 (BT left 1).
- t = 4: All available → Draw = P2 → run P2 (BT left 2).
- t = 5: All available → Draw = P2 → run P2 (BT left 1).
- t = 6: All available → Draw = P4 → run P4 (BT left 0, finishes).
- t = 7: All available except P4 → Draw = P2 → run P2 (BT left 0, finishes).
- t = 8: P1 and P3 remain → Draw = P1 → run P1 (BT left 0, finishes).
- t = 9: Only P3 remains → Draw = P3 → run P3 (BT left 0, finishes).

Since the process runs for 1 time unit after winning the draw, the completion time is slot-index + 1.

P1 finishes at time 9.
P2 finishes at time 8.
P3 finishes at time 10.
P4 finishes at time 7.

Grading guide: 1 point each for expanding/listing the arrival times and burst time (10 points). 0.5 point each for getting the completion time correct for each of the 4 processes (total 2 points).

(b) (10 points) Average turnaround time and the per process turnaround time

Turnaround time = completion time - arrival time

P1 $= 9 - 0 = 9$
P2 $= 8 - 1 = 7$
P3 $= 10 - 2 = 8$
P4 $= 7 - 3 = 4$

Average turnaround time = (9+7+8+4)/4 = 28/4 = 7

Grading guide: There are 5 components (4 for each process and ATT). Each component has a formula and the answer. 1 point for (using) the formula and 1 point for the answer. This would give a maximum of 10 points.