

Mem & OS Review

COSC 208, Introduction to Computer Systems, 2021-11-19

Announcements

- Project 3 due Thursday, December 2

Outline

- Memory hierarchy
- Caching
- Processes
- Scheduling
- Threads

Memory hierarchy

- Q1: What is the **fastest volatile** memory? — registers
- Q2: What is the **fastest non-volatile** memory? — solid state drive (SSD)
- Q3: Why is a hard disk drive (HDD) slower than a solid state drive (SSD)? — an HDD has moving parts that need to be moved into position before data can be read/written
- Q4: Why is accessing main memory (i.e., Random Access Memory (RAM)) slower than accessing a cache? — a cache is part of the CPU, but data needs to travel across a bus to move between the CPU and main memory
- Q5: Why do solid state drives (SSDs) cost less per unit of capacity than main memory (i.e., Random Access Memory (RAM))? — SSDs have slower access latency

Caching

- Q6: Assume the cache size is 3 and the **optimal** cache replacement algorithm is used. Indicate what happens with the cache on each data access.
 - Access 2 — +2
 - Access 4 — +4
 - Access 1 — +1
 - Access 2 — Hit
 - Access 4 — Hit
 - Access 3 — -1/+3
 - Access 2 — Hit
 - Access 4 — Hit
 - Access 1 — -3/+1
 - Access 2 — Hit
 - Access 4 — Hit
 - Access 1 — Hit
- Q7: Assume the cache size is 3 and the **least recently used (LRU)** cache replacement algorithm is used. Indicate what happens with the cache on each data access.
 - Access 2 — +2
 - Access 4 — +4
 - Access 1 — +1
 - Access 2 — Hit
 - Access 4 — Hit
 - Access 3 — -1/+3
 - Access 2 — Hit
 - Access 4 — Hit
 - Access 1 — -3/+1
 - Access 2 — Hit
 - Access 4 — Hit

- Access 1 — Hit

Processes

- Q8: Write a program that creates a new process. The child process should print "I am a child"; the parent process should print "I am a parent; my child is CPID" (replacing CPID with the child's PID).

```
int main() {
    int pid = fork();
    if (pid == 0) {
        printf("I am a child\n");
    }
    else {
        printf("I am a parent; my child is %d\n", pid);
    }
}
```

- Q9: Will the output produced by your program always appear in a particular order? Why or why not?
 - No, because the parent does not **wait** for the child to finish before printing, and the OS scheduler determines which order the processes run
 - If you included a call to **wait** before the call to **printf** in the else body, then the answer would be yes, because the parent waits for the child to finish before printing

Scheduling

Consider the following set of processes:

| Process | Duration | Arrival Time |
|---------|----------|--------------|
| A | 20 | 0 |
| B | 15 | 0 |
| C | 25 | 5 |
| D | 5 | 10 |

- Q10: Draw the schedule when a First In First Out (FIFO) scheduling algorithm is used.



- Q11: Compute the turnaround and wait time for each process based on the above schedule.

| Process | Turnaround | Wait |
|---------|------------|------|
| A | 20 | 0 |
| B | 35 | 20 |
| C | 60 | 30 |
| D | 55 | 50 |

- Q12: Draw the schedule when a Shortest Job First (SJF) scheduling algorithm is used.



- Q13: Compute the turnaround and wait time for each process based on the above schedule.

| Process | Turnaround | Wait |
|---------|------------|------|
| A | 40 | 20 |
| B | 15 | 0 |
| C | 55 | 35 |
| D | 10 | 5 |

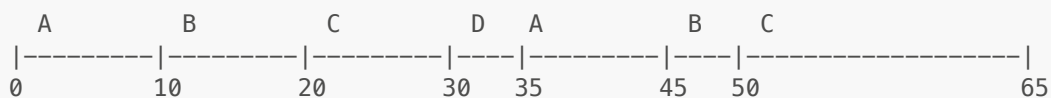
- Q14: Draw the schedule when a Shortest Time to Completion First (STCF) scheduling algorithm is used.



- Q15: Compute the turnaround and wait time for each process based on the above schedule.

| Process | Turnaround | Wait |
|---------|------------|------|
| A | 40 | 20 |
| B | 15 | 0 |
| C | 60 | 35 |
| D | 10 | 5 |

- Q16: Draw the schedule when a Round Round (RR) scheduling algorithm is used with a time quantum of 10.



- Q17: Compute the turnaround and wait time for each process based on the above schedule.

| Process | Turnaround | Wait |
|---------|------------|------|
| A | 45 | 25 |
| B | 50 | 35 |
| C | 60 | 35 |
| D | 25 | 20 |

Threads

A program contains the following functions:

```

void *dec(void *arg) {
    int *t = (int *)arg;
    *t--;
}

void *inc(void *arg) {
    int *t = (int *)arg;
    *t++;
}

void *zero(void *arg) {
    int *t = (int *)arg;
    *t = 0;
}

```

For each of the following main methods, list **all possible outputs** the program could produce. Assume threads are only preempted if they become blocked waiting for other threads.

- Q18:

```

int main() {
    int *total = malloc(sizeof(int));
    *total = 2;
    pthread_t thrA, thrB;
    pthread_create(&thrA, NULL, &inc, total);
    pthread_create(&thrB, NULL, &inc, total);
    pthread_join(&thrA);
    pthread_join(&thrB);
    printf("%d\n", *total);
}

```

- 4

- Q19:

```

int main() {
    int *total = malloc(sizeof(int));
    *total = 2;
    pthread_t thrA, thrB;
    pthread_create(&thrA, NULL, &dec, total);
    pthread_create(&thrB, NULL, &zero, total);
    pthread_join(&thrA);
    pthread_join(&thrB);
    printf("%d\n", *total);
}

```

Possible outputs:

- 0
- -1

- Q20:

```

int main() {
    int *total = malloc(sizeof(int));
    *total = 2;
}

```

```
pthread_t thrA, thrB;
pthread_create(&thrA, NULL, &zero, total);
pthread_join(&thrA);
pthread_create(&thrB, NULL, &inc, total);
pthread_join(&thrB);
printf("%d\n", *total);
}
```

Possible outputs:

- 1