Mustafa Al Awadi

ID: 40217764

Comp346

2024-10-28

Theory Assignment 3

1.

i. a) A relocatable program is a program that can execute without modification regardless of the address from the memory.

b) What makes a program relocatable is whether the data has a relative address or not. If it does, then it is relocatable, however, in case it is absolute address then it can't be changed and therefore won't be able to relocate.

c) Programs need to be relocatable because otherwise programs would be running in the same memory location which can cause execution of programs to slow down. In such case, they must locate an open memory address to execute there.

ii. In small page sizes, there could be less wasted memory in comparison to large page size. Waste of memory, especially in large page size, can lead to internal fragmentation. There can be more page faults in smaller page size in general compared to larger page sizes.

iii. There won't be external fragmentation, and we can swap easily between disk and memory as frames and pages hold the same sizes.

iv. There will be a decrease in memory space occupied in the segmentation mechanism and also there will be no internal fragmentation.

2.

a) The critical sections inside the wait and signal operations are :

Critical section for wait():

```
wait () {
    Disable interrupts;
    sem.value--;
    if (sem.value < 0) {
        save_state (current) ; // current process
        State[current] = Blocked; //A gets blocked
        Enqueue(current, sem.queue);
        current = select_from_ready_queue();
        State[current] = Running;
        restore_state (current);  //B starts running
    }
    Enable interrupts;
}
```

sem.value-- decrementing the semaphore value is critical because it modifies the shared semaphore counter.

if (sem.value < 0) { ... } Checking the semaphore value and enqueuing the process if it is negative.

Critical section for signal():

```
signal(){
   Disable interrupts;
    sem.value++;
    if (sem.value <= 0){
      k = Dequeue(sem.queue);
      State[k] = Ready;
      Enqueue (k, ReadyQueue);
    }
    Enable interrupts;
}
```

sem.value++ Incrementing the semaphore value modifies the shared counter.

if (sem.value <= 0) { ... } Checking the semaphore value and dequeuing a process from the semaphore queue to add it to the ready queue.

b) If sem.value = 1, the wait() method will execute as follows: the if condition (sem.value < 0) will not be true, so it will skip that block and proceed to re-enable interrupts. After completing wait(), the signal() method may be called.

In signal(), interrupts are first disabled, and sem.value is incremented to 2. Since the if condition (sem.value <= 0) is not satisfied, the block inside the condition is skipped, and interrupts are re-enabled. As a result, the signal() operation simply increases sem.value without actually affecting the wait() operation or resetting the semaphore value to 0, potentially leading to unexpected behavior.

c) No, it will not execute. Interrupts need to be controlled at the hardware level to ensure the operation is atomic. If process B starts with interrupts disabled, synchronization cannot be guaranteed, and the operation will lose its atomicity.

3.

TLB time is 0.2 microseconds, single memory access time is 1 microsecond, disk time is 20000 microseconds , page fault rate is 2% and the TLB hit rate is 80%.

Case 1: TLB hit (T1)

If the required page is found in the TLB (80% of the time):

T1=(TLB hit rate)×(TLB access time + memory access time)T1=(TLB hit rate)×(TLB access time + memory access time)

T1=0.8×(0.2+1)=0.96 microsecondsT1=0.8×(0.2+1)=0.96microseconds

Case 2: TLB miss but memory hit (T2)

If the page is not in the TLB but is found in memory (98% of page accesses, with 20% of these being TLB misses):

T2=(Page hit rate)×(TLB miss rate)×(TLB time + 2 memory accesses)T2=(Page hit rate)×(TLB miss rate)×(TLB time + 2 memory accesses)

T2=0.98×0.2×(0.2+1+1)=0.4312 microsecondsT2=0.98×0.2×(0.2+1+1)=0.4312microseconds

Case 3: TLB miss, memory miss, disk hit (T3)

If the page is not in the TLB or memory, resulting in a page fault (2% of accesses, with 20% of these being TLB misses):

T3=(Page fault rate)×(TLB miss rate)×(Disk time + TLB time + 2 memory accesses)T3=(Page fault rate)×(TLB miss rate)×(Disk time + TLB time + 2 memory accesses)

T3=0.02×0.2×(20,000+0.2+1+1)=0.02×0.2×20,002.2=80.0088 microsecondsT3=0.02×0.2×(20,000+0.2+1+1)=0.02×0.2×20,002.2=80.0088microseconds

The total effective memory access time is the sum of the contributions from all three cases:

EMAT=T1+T2+T3EMAT=T1+T2+T3

EMAT=0.96+0.4312+80.0088=81.4 microsecondsEMAT=0.96+0.4312+80.0088=81.4microseco

nds

4.

a) LRU:

	0	1	2	0	1	2	0	1	2	3	6	7	6	7	0	1	2	3	4
F1	0	0	0	0	0	0	0	0	0	3	3	3	3	3	0	0	0	3	3
F2		1	1	1	1	1	1	1	1	1	6	6	6	6	6	1	1	1	4
F3			2	2	2	2	2	2	2	2	2	7	7	7	7	7	2	2	2

There are 11 page faults on the table which are highlighted.

b) Belady:

	0	1	2	0	1	2	0	1	2	3	6	7	6	7	0	1	2	3	4
F1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
F2		1	1	1	1	1	1	1	1	1	1	7	7	7	7	7	2	2	2
F3			2	2	2	2	2	2	2	3	6	6	6	6	6	6	6	3	4

There are 10 page faults on the table which are highlighted.

c) Working set:

	0	1	2	0	1	2	0	1	2	3	6	7	6	7	0	1	2	3	4
F1	0	0	0	0	0	0	0	0	0	3	3	3	3	3	0	0	0	3	3
F2		1	1	1	1	1	1	1	1	1	6	6	6	6	6	1	1	1	4
F3			2	2	2	2	2	2	2	2	2	7	7	7	7	7	2	2	2
	{0}	{0,1,2}	{0,1,2}	{0,1,2}	{0,1,2}	{0,1,2}	{0,1,2}	{0,1,2}	{0,1,2}	{1,2,3}	{2,3,6}	{3,6,7}	{3,6,7}	{0,6,7}	{0,6,7}	{0,1,7}	{0,1,2}	{3,1,2}	{3,4,2}

Ther are 11 page faults shown in the table which are highlighted.

5.

a) The main advantage of this system is the significant difference in access time, as CPU memory can be accessed much more quickly.

b) The drawback of this system is its memory inefficiency, as the page table requires more memory than what is needed to store the data itself.

6.

i. The global replacement algorithm allows selecting any page in memory for replacement, unlike the local replacement algorithm, which restricts selection to pages associated with the same process.

ii. The primary drawback of the global replacement algorithm is its lack of scalability, which can lead to poor system performance. This may cause process duplication, further slowing down memory operations.

7.

i. If TPF > TFS, we would experience fewer page faults than service page faults, resulting in optimal multiprogramming performance.

ii. If TPF < TFS, we would encounter more page faults than service page faults, leading to the poorest multiprogramming performance.

iii. If TPF = TFS, the system would be relatively stable, as the number of page faults and service page faults would be close to each other.

8.

Advantages: There is no need to manually close the file using the .close() method.

Disadvantages: Since it bypasses File I/O, it can lead to privacy leaks and potential deadlocks, as it will always wait for the file to be closed.

9.

a) Pre-emptive scheduling occurs when a process transitions from the running state to the ready state, or more specifically, from waiting to ready. In contrast, non-pre-emptive scheduling happens when a process moves from the running state to the waiting state. Strictly using non-pre-emptive scheduling would eliminate priority management and context switching, making it an impractical option.

b) A small quantum size would result in more frequent context switching, significantly slowing down the system and reducing overall performance. On the other hand, a large quantum size would lead to fewer context switches, but response times would still increase because FIFO scheduling would be used, which is not ideal for efficient multiprogramming.

10.

One advantage of this queue system is that processes can move between queues. Additionally, higher-priority queues are allocated more CPU time. Processes that consume more time will have their priority reduced, allowing other processes to run. This helps avoid starvation, but it may lead to fairness issues, as low-priority processes might end up running at the same time as high-priority ones.

11.

a) FCFS scheduling:

P0 (20)	P1 (35)	P2 (56)	P3 (63)	P4 (75)

b) Non-pre-emptive SJF scheduling:

P3 (7)	P4 (19)	P1 (34)	P0 {54}	P2 (75)

c) Non-preemptive priority scheduling:

P1 (15)	P4 (27)	P0 (47)	P2 (68)	P3 (75)

d) Pure Round-Robin scheduling with the quantum = 3:

Process	Service Time	Highest Multiple of 3	Remainder
P0	20	72	0
P1	15	61	3
P2	21	75	6
P3	7	40	9
P4	12	55	12

P0 (3)	PI (6)	P2 (9)	P3 (12)	P4 (15)	P0 (18)	P1 (21)	P2 (24)	P2 (27)

P0: 6, 14 remaining

P1: 6, 9 remaining

P2: 6, 15 remaining

P3: 6, 1 remaining

P4: 3, 9 remaining

P4 (30)	P0 (33)	P1 (36)	P2 (39)	P4 (42)	P0 (45)	P1 (48)	P2 (51)	P4 (54)

P0: 12, 8 remaining

P1: 12, 3 remaining

P2: 12 ,9 remaining

P3: 6, 1 remaining

P4: 9, 0 remaining

We notice that process P4 has been used to it's maximum time

P0 (57)	P1 (60)	P2 (63)	P0 (66)	P2 (69)	P0 (71)	P3 (72)	P2 (75)

P0: 20, 0 remaining

P1: 15, 0 remaining

P2: 21 ,0 remaining

P3: 7, 0 remaining

P4: 9, 0 remaining

All the processes have reached their maximum time

b)

Waiting Time	P0	P1	P2	P3	P4
FCFS	0	20	35	56	63
SJF	34	19	54	0	7
Priority Scheduling	27	0	47	68	15
Round-Robin	69	57	72	71	51

c)

Waiting Time	P0	P1	P2	P3	P4
FCFS	0	20	35	56	63
SJF	34	19	54	0	7
Priority Scheduling	27	68	47	75	15
Round-Robin	0	3	6	9	12

d)

Waiting Time	P0	P1	P2	P3	P4
FCFS	20	35	56	63	75
SJF	54	34	75	7	19

Priority Scheduling	47	15	68	75	27
Round-Robin	71	60	75	72	54