COSC 450 Operating System Midterm #1

10/17/2024

Name:_____.

1. Short answer questions

   a. **(2 pt.) What is a translation look-aside buffer (TLB) and how does it solve the problem of slow memory access?**
   Answer) The TLB is a small, fast-lookup hardware cache that stores a few entries from the page table. It improves memory access times by allowing the memory management unit (MMU) to check the TLB first. If the page number is found in the TLB (TLB hit), the frame number is available immediately, avoiding the need for two memory accesses.

   b. **(2 pt.) Why does memory access slow down by a factor of 2 when the page table is maintained in memory?**
   Answer: Memory access slows down by a factor of 2 because the system first needs to access memory to retrieve the page frame number from the page table, and then it has to access memory again using the physical address (page frame number + offset).

   c. **(2 pt.) What is the main purpose of the multilevel page table method?**

   Answer) The main purpose of the multilevel page table method is to avoid keeping all the page tables in memory at once, reducing the memory overhead associated with large page tables.

   d. **(2 pt.) What problem does a hashed page table aim to solve when managing larger virtual address spaces?**
   Answer) A hashed page table aims to efficiently manage larger address spaces by reducing the number of entries in the page table and minimizing memory overhead.

   e. **(2 pt.) How does the number of page table entries in the inverted page table relate to physical memory?**

   Answer) The number of page table entries in the inverted page table is equal to the number of page frames in physical memory.

2. **(10 pt.) A computer system generates a 32-bit virtual address for a process. This system has 16 GB RAM and page size is 2KB.**
   a. **If each entry in the page table needs 64 bits per entry, calculate <u>the maximum possible</u> <u>size of the page table by KB</u>.**
   Sol) Possible maximum size of virtual space = $2^{32} = 2^{22} \times 2^{10} = 2^{22}$ **KB**

   ∴Maximum # of pages per a process = virtual space / a page size = $2^{22}/2 = 2^{21}$ pages .

   Since maximum number of pages is 2^{21} pages, there are 2^{21} possible entries in a page table.
   Maximum size of page table per a process = number of pages × one entry size

   = $2^{21} \times 64$ bits = $(2^{21} \times 64)$ /8 Byte= $2^{21} \times 8$ byte= **16 MB**

   b. Page frame number information for each page must be saved in the page table. <u>How many bits</u> does it need to save page frame number information in a page table entry?
   Sol) simply need calculate the number of page frames

   # of page frame = size of RAM / size of page = 16GB / 2KB = $16 \times 2^{30} / 2 \times 2^{10} = 2^{23}$ page frames ∴ **23 bits for page frame number.**

3. **(10 pt.) Probabilistic model for multiprogramming.**

   a) Lets *p* is an average fraction of time a process is waiting for I/O. If there are n processes in the memory at once, what will be the CPU utilization?
   $1 - p^n$

   b) A computer has 10 GB of memory, with operating system taking up 2GB and each program taking up 1 GB. With an 85 % average I/O wait, calculate CPU utilization

   n= (10 GB – 2 GB)/1GB = 8
   p= 0.85 CPU utilization =1- $(0.85)^8$ = 0.7275 about 73%

   c) By doubling the size of memory, what will be the CPU utilization?

   n = (20 GB – 2GB) / 1GB = 18, p = 0.85 $1 - (0.85)^{18}$ = 0.94 about 94 %

4. **(10 pt.) A Computer with a 64 bit virtual address use two-level page table. Virtual addresses are split into a 30 bit top-level page table field, a 20 bit second-level page table field and an offset.**

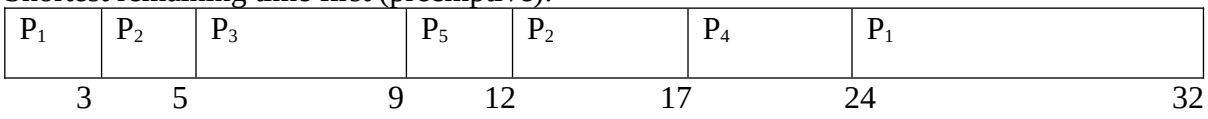
   a. **What is the size of each page ?**
      2^{14} = 16 KB
   b. **How many possible page tables are there?**
      2^{30} page tables +1
   c. **How many possible pages are there?**
      2^{30} x 2^{20} pages = 2^{50} pages
   d. **If the system supports 16GB RAM, how many bits need to be reserved for saving page frame number in the each of page table entry?**
      Sol) There are 16GB /16 KB = 16 $\times$ 2^{30} / 16 $\times$ 2^{10} = 2^{20} page frames. The system need reserve 20 bit for page frame number.

5. **(10 pt.) Consider the following set of processes (each processes are 100 % CPU-bounded).**

Process	CPU-Time	Priorities	Arrival Time
P_1	11	2	0
P_2	7	3	3
P_3	4	5	5
P_4	7	4	7
P_5	3	1	8

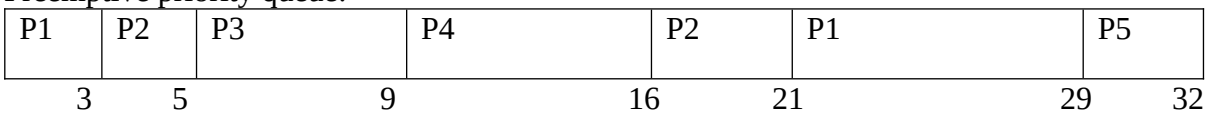
Calculate average waiting time and average turnaround time with Shortest Remaining Time First and Preemptive Priority Queue process scheduling algorithms.

Shortest remaining time first (preemptive):



- Average waiting time – ((24-3)+(12-5)+0+(17-7)+(9-8))/5 =(21 + 7 + 0 + 10 + 1)/5 = 7.8
- Average turnaround time – ((32-0)+(17-3)+(9-5)+(24-7)+(12-8)/5= (32 + 14 + 4+ 17 + 4)/5 = 14.2

Preemptive priority queue:



- Average waiting time – (18+11+ 0+ 2+ 21)/5 =10.4
- Average turnaround time – (29+ 18+ 4+ 9+ 24)/5 = 16.8

6. (10 pt.) Mr. Computer tries to solve the race condition problem in the producer-consumer problem by using semaphores. He designs the following solution that utilizes three semaphores: mutex for mutual exclusion, empty to track the number of empty spaces in the buffer, and full to count the number of available items in the buffer. However, his solution results in a deadlock, causing both the producer and consumer to enter an indefinite sleep state. Describe a scenario that leads to this deadlock in his solution.

```
#define N 100
typedef int semaphore;
semaphore mutex = 1;
semaphore empty = N;
semaphore full = 0;

void producer ()
{
        int item;
        while (ture)
        {
            item = produce_item();
            down (&empty);
            down (&mutex);
            insert_item(item);
            up(&mutex);
            up(&full);
        }
}
```

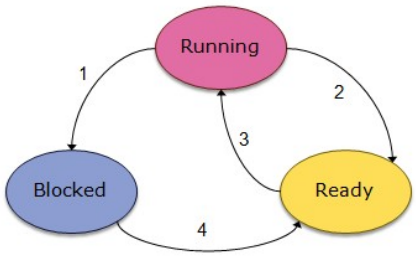
```
void consumer()
{
        int item;

        while (true)
        {
            down(&mutex);
            down(&full);
            item = remove_item();
            up(&mutex);
            up(&empty);
            consume_item(item);
        }
}
```

Sol) Let's assume the following values at time t: mutex = 1, full = 0, and the CPU scheduler selects the consumer.

- The consumer performs a down operation on mutex (making mutex = 0).
- The consumer then attempts a down operation on full. Since full = 0, the consumer goes to sleep, waiting in the queue for full.
- The CPU scheduler selects the producer.
- The producer performs a down operation on empty (making empty = N - 1).
- The producer then attempts a down operation on mutex. Since mutex = 0, the producer also goes to sleep, waiting in the queue for mutex.
- At this point, both the consumer and the producer are asleep, leading to a deadlock.

7. (5 pt.) A process is stayed in one of three states: running, blocked or ready state. Briefly discuss each of states and transaction between states.

- Running state – a process using CPU
- Ready state – a process waiting for CPU
- Blocked state – a process waiting for I/O finish
- Transaction 1 – a process need I/O
- Transaction 2 – a process time out
- Transaction 3 – scheduler select a process to run
- Transaction 4 – a process finish I/O and ready to run



8. **(10 pt.) You are going to compare the storage space needed to keep track of free memory using a bitmap versus using a linked list. The 2GB memory is allocated in units of 4KB. For the free list, let's assume that memory is currently consists of an alternating sequence of segment and holes, each 64 KB. Also assume that each node in the linked list needs a 32-bit memory address, a 16-bit length, and a 16-bit next node field. How many bytes of storage are required for each method?**

Sol)

- Bitmap: #of allocation unit = 2GB/4KB = $(2 \times 2^{30})/(4 \times 2^{10}) = 2^{31}/2^{12} = 2^{19}$ units
  Size of the bitmap = 2^{19} bits = **2^{16} byte**

- The linked list: number of node for linked list= 2 GB /64KB = $2^{31}/2^{16}$ or 2^{15} nodes.
  size of each node = 32+16+16 = 64 bit = 8 byte =2^3 bytes
  Total size of linked list = number of node $\times$ size of a node = $2^{15} \times 2^3$ bytes = **2^{18} bytes**.

9. (10 pt.) A computer can generate 32 bit virtual address for a process. This system has 4GB RAM and page size is 4KB.
   a. Let's assume page #0 is map to page frame #23
   b. Let's assume page #1 is map to page from #5
   c. Let's assume page #2 is map to page frame #1100
   d. Let's assume page #3 is map to page frame #1230
   e. Let's assume page #4 is map to page frame #135
   f. Let's assume page #5 is map to page frame #95
   g. Let's assume page #11 is map to page frame #7.

Calculate physical address for each of following virtual addresses

a) 19845
   virtual address $19845/(4 \times 2^{10}) = 4.84$ in page #4 (map to page frame #135).
   virtual page #4 begin with address $4 \times 4 \times 2^{10}$
   page frame #135 is start with address $135 \times 4 \times 2^{10}$

physical address = $135 \times 4 \times 2^{10}$ + (19845 - $4 \times 4 \times 2^{10}$) = 552,960 + 3,461 = 556,421

b) 21581
virtual address 21581/(4×2^{10})= 5.26 in page #5

virtual page #5 begin with address $5 \times 4 \times 2^{10}$
page frame #95 is start with address $95 \times 4 \times 2^{10}$
physical address = $95 \times 4 \times 2^{10}$+ (21581 - $5 \times 4 \times 2^{10}$) = 389120 + 1101 = 390221

10. (10 pt.)Let's assume there are two processes P_0 and P_1 sharing a resource in the critical section. The following shows a solution for the race condition with busy waiting. Variable **Permit** can be 0 or 1. If **Permit** = 0, only process P_0 can go to the critical section. Once P_0 finish its job in the critical section, P_0 set **Permit** = 1, let process P_1 enter critical section.
If **Permit** = 1, only process P_1 can go to the critical section. Once P_1 finish its job in the critical section, P_1 set **Permit** = 0, let process P_0 enter critical section.

```
repeat
      while Permit ≠ i do
            ; (no-operation)
      Critical Section
      Permit = j;

      Remainder Section
until false
```

Show this solution cannot solve race condition.

sol)
lets assume Permit = 0 at time T
P_0 tries to enter C.S. and can enter since Permit =0.
P_0 finish its job in C.S. and set Permit =1
P_1 is currently running outside C.S ,it is terminated with fatal error.
P_0 tries to enter C.S. again but P_0 never can.

11. (5 pt.) Multiple jobs can run in parallel and finish faster than if they had run sequentially. Suppose that two jobs, each needing 20 minutes of CPU time, start simultaneously. How long will the last one take to complete if they run sequentially? How long if they run in parallel? Assume 50% I/O wait.

Sol) If each job has 50% I/O wait, then it will take 40 minutes to complete in the absence of competition. If run sequentially, the second one will finish 80 minutes after the first one starts. With two jobs, the approximate CPU utilization is $1 - 0.5^2 = 0.75$. Thus, each one gets 0.375 CPU minute per minute of real time. To accumulate 20 minutes of CPU time, a job must run for20/0.75 +20/0.75 minutes, or about 53.33 minutes. Thus running sequentially the jobs finish after 80 minutes, but running in parallel they finish after 53.33 minutes.