**CSE 2431**  **HOMEWORK 2**

**SP 23**

**NAME: Jiahao Zhang (zhang.12789)**

**Due:** **Tuesday, February 14th, 2023 by 11:30 pm on Carmen (as long as you submit before midnight, the assignment will be counted as on time, but at midnight or after (even 1 second!), it will be counted as late)**

**NOTE: *Please keep a copy of your homework to study for the exam.***

1. **CPU Scheduling**

Analyze scheduling algorithms for the following five processes given the process CPU burst time, and arrival time.

|  |  |  |
| --- | --- | --- |
| **Processes** | **CPU Burst (ms)** | **Arrival Time** |
| P0 | 9 | 0 |
| P1 | 4 | 2 |
| P2 | 7 | 4 |
| P3 | 2 | 6 |
| P4 | 11 | 8 |

For each of the following scheduling algorithms (i) Draw a Gantt diagram (you can use a one row table with an appropriate number of columns); (ii) Calculate clearly (show how you do the calculation; see the class slides on Carmen for formulas) the total wait time for each process; and (iii) Calculate clearly (show how you do the calculation) the average wait time for all processes. **NOTE:** If the scheduling algorithm cannot uniquely determine which of 2 or more processes should be scheduled, the process among the alternatives which has the earliest arrival time should be scheduled.

1. **FIFO/First Come First Served (FCFS)** (Arrival times should be considered.)

(i) Gantt chart (Example given below, but ***it is not correct for this problem***; you can add columns if needed, and you edit the table by adding or deleting columns, and changes the process in each box; you can also move the lines dividing the columns; make other Gantt charts below this way)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **P0** | **P1** | **P2** | **P3** | **P4** |

0  **9 13 20 22 33**

(ii) Total wait time per process

Wait\_time = completion\_time – arrival\_time – burst\_time

P0: 9 - 0 = 9

P1: 13 - 2 = 11

P2: 20 - 4 = 16

P3: 22 - 6 = 16

P4: 33 - 8 = 25

(iii) Average wait time per process

(9 + 11 + 16 + 16 + 25) / 5 = 15.4

1. **(Non-preemptive) Shortest Job First**

(i) Gantt chart (you can add columns if needed; make other Gantt charts below this way)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **P0** | **P3** | **P1** | **P2** | **P4** |

0  **9 11 15 22 33**

(ii) Total wait time per process

P0: 9 - 0 = 9

P1: 15 - 2 = 13

P2: 22 - 4 = 18

P3: 11 - 6 = 5

P4: 33 - 8 = 25

(iii) Average wait time per process

(9 + 13 + 18 + 5 + 25) / 5 = 14

1. **(Preemptive) STCF**

(i) Gantt chart (you can add columns if needed)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **P0** | **P1** | **P3** | **P0** | **P2** | **P4** |

0 **2 6 8 15 22 33**

(ii) Total wait time per process

P0: 15 - 0 = 15

P1: 6 - 2 = 4

P2: 22 - 4 = 18

P3: 8 - 6 = 2

P4: 33 - 8 = 25

(iii) Average wait time per process

(15 + 4 + 18 + 2 + 25) / 5 = 12.8

1. **Round Robin Scheduling** (Assume that the time quantum is 3 ms). Also assume that if a new process arrives at exactly the same time as the time slice of the executing process expires the executing process is put at the end of the ready queue *after* the arriving process.

(i) Gantt chart (you can add columns if needed)

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| P0 | P1 | P0 | P2 | P3 | P1 | P4 | P0 | P2 | P4 | P2 | P4 |

0 3 6 9 12 14 15 18 21 24 27 28 33

(ii) Total wait time per process

P0: 21 - 0 = 21

P1: 15 - 2 = 13

P2: 28 - 4 = 24

P3: 14 - 6 = 8

P4: 33 - 8 = 25

(iii) Average wait time per process

(21 + 13 + 24 + 8 + 25) / 5 = 18.2

**2. Addressing for Paged Memory**

Complete the table below by filling in the empty boxes with the correct value for number of virtual address bits; number of virtual page bits; number of page offset bits; number of virtual pages; and number of bytes in each page.

For page size, express the value in KB (see note below). For number of virtual pages, write the value as a power of 2.

NOTE: Assume 1 KB = 1024 bytes (210 bytes)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Virtual Address Bits | Virtual Page Bits | Page Offset Bits | No. Virtual Pages | No. Bytes per Page |
| 32 | **22** | 10 | **2^22** | **1KB** |
| 38 | 26 | **12** | **2^26** | **4KB** |
| 20 | **10** | **10** | 210 | **1KB** |
| 30 | **19** | **11** | **2^19** | 2 KB |
| 48 | **33** | 15 | **2^33** | **32KB** |

**3. Virtual memory**

Fill in the boxes in the table below for the approaches to virtual memory discussed in class, with T, if the characteristic applies to the approach, or F if it does not.

**Characteristics**

1. No MMU hardware is needed to implement the approach (no MMU or registers in an MMU).
2. The whole address space for the process must be loaded into a contiguous sequence of bytes in memory in order for the process to run.
3. The whole address space of the process must be loaded into memory (but not necessarily in a contiguous sequence of bytes) in order for the process to run.
4. The whole process, or parts of it, can be dynamically relocated by the OS while the process is in execution (that is, between the time it is created and the time it terminates execution).
5. Each segment of the process must be loaded into a contiguous sequence of bytes in memory.
6. No general mechanism for protecting processes from each other (i.e., preventing any process from accessing another process’s address space) is provided.
7. No mechanism for processes to share a portion of their address spaces (but not their entire address space) is provided.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Approach** | **Characteristic** | A | B | C | D | E | F | G |
| Static relocation | X | T | T | T | F | T | T | T |
| Dynamic Relocation with Base | X | F | T | T | T | T | T | T |
| Dynamic Relocation with Base + bounds | X | F | T | T | T | T | F | T |
| Segmentation | X | F | F | T | T | T | F | F |
| Paging | X | F | F | F | T | F | F | F |

**4. Virtual Memory – Segmentation**

1. Suppose the use of segmentation and an 18 bit virtual address space for a program running as a process with (a maximum of) 4 segments.
2. Here is the segment table

|  |  |  |  |
| --- | --- | --- | --- |
| Segment | Base | Bounds | RW (Protection bits) |
| 0 | 0xc000 | 0xffff | 10 |
| 1 | 0x8000 | 0x0fff | 11 |
| 2 | 0x1000 | 0x07ff | 11 |
| 3 | 0x4000 | 0x0dff | 11 |

1. Translate the following virtual addresses in the program to the corresponding physical addresses.
   1. ***Show clearly how you calculated the physical address***.
   2. Say whether the memory reference results in a segmentation fault (invalid memory access), and ***if so, say why clearly***.

Virtual Addresses

1. Read 0x100e8 :

page 1, offset e8

=> 0x8000+0xe8

= 0x80e8

* + - 1. Write 0x30c00

page 3, offset c00

=> 0x4000 + 0xc00

= 0x4c00

* + - 1. Write 0x20a04

page 2, offset a04

=> page fault because 0xa04>the bound 0x7ff

* + - 1. Write 0x02bfe

page 0, offset 2bfe

=> 0xc000 + 0x2bfe

= 0xebfe

***The homework must be typed, including diagrams, then printed out and stapled before submission.***