|  |  |
| --- | --- |
| Problem Set 1: Week 1 ~ 7 | |
| Student ID: 21102376 | Student Name: 이연우 |

1. Define the operating system, and describe the main purposes of an operating system?

**An operating system (OS) is a software program that manages computer hardware resources and provides common services for computer programs. The main purposes of an operating system are to provide an interface between the computer hardware and the user, to manage and allocate computer resources such as CPU time, memory, and input/output devices, and to provide services such as file management, process management, and security. In essence, an operating system acts as a platform for other software programs to run on, and it provides an environment in which users can perform tasks efficiently and securely. Without an operating system, a computer would be unable to function effectively, as there would be no way for software programs to communicate with the hardware.**

2. Represent the decimal number 63.25 in IEEE 754 standard single precision format

**The decimal number 63.25 in IEEE 754 standard single precision format is 0 10000101 11111010000000000000000.**

3. In RISC-V assembly, write an assembly language version of the following C code segment:

for (i = 0; I < 98; i++) {

C[i] = A[i+1] – A[i] + B[i+2]

}

The start addresses of arrays A, B, and C are stored in x18, x19, x20, respectively. The local variable i is stored in x5.

**loop:**

**bit x5, 98, loop\_end # if I >= 98, exit loop**

**addi x6, x18, 4(x5) # A[i+1]**

**lw x6, 0(x6) # load A[i+1]**

**lw x7, 0(x18)(x5) # load A[i]**

**addi x8, x19, 8(x5) # B[i+2]**

**lw x8, 0(x8) # load B[i+2]**

**sub x9, x6, x7 # A[i+1] – A[i]**

**add x10, x9, x8 # A[i+1] – A[i] + B[i+2]**

**addi x11, x20, 4(x5) # &C[i]**

**sw x10, 0(x11) # store C[i]**

**addi x5, x5, 1 # I++**

**j loop # jump back to loop**

**loop\_end:**

4. Describe each of the four segments of the memory layout of process: text, data, heap, and stack.

1. **Text segment: The text segment contains the executable code of the process. It is also called the code segment or the program segment. The text segment is typically read-only and includes instructions that the processor can execute.**
2. **Data segment: The data segment contains the initialized and uninitialized global and static data of the process. This includes global variables and static variables. The initialized data is stored in a read-write area, while the uninitialized data is stored in a read-write but zero-initialized area.**
3. **Heap segment: The heap segment contains the dynamically allocated memory of the process. This is memory that is allocated at runtime by calling functions like malloc(), calloc(), and realloc(). The heap grows and shrinks dynamically during the execution of the process, depending on the amount of memory requested and released.**
4. **Stack segment: The stack segment contains the stack of the process. The stack is used to store local variables, function arguments, return addresses, and other data that is specific to each function call. The stack grows and shrinks dynamically during the execution of the process, depending on the number of function calls and returns. The stack is typically located at the high end of the memory and grows downwards.**

5. True or False

A. The unix exec system call creates a new process. **: False**

B. Processes may use the same virtual address but will actually access different physical addresses. **: True**

C. In a one-level paging scheme, a single page table and a single TLB are shared by all processes. **: True**

D. FCFS has throughput at least as good as RR. **: False**

6. How many processes are created? **: The program creates 8 processes.**

# include <stdio.h>

# include <unistd.h>

int main(void)

{

fork();

fork();

fork();

return 0;  
}

7. Explain the difference between preemptive and nonpreemptive scheduling

**Preemptive and nonpreemptive scheduling are two types of scheduling algorithms used by the operating system to manage the allocation of CPU time among multiple processes.**

**Nonpreemptive scheduling, also known as cooperative scheduling, is an algorithm in which a process voluntarily gives up the CPU after completing its task or when it requests a service that cannot be provided immediately. In nonpreemptive scheduling, once a process gets the CPU, it keeps the CPU until it releases it or enters a waiting state. This approach ensures that the process running on the CPU is in control of the CPU for the duration of its time slice. Nonpreemptive scheduling is simpler to implement but can cause a problem of starvation if one or more processes dominate the CPU.**

8. Suppose that the following processes arrive for execution at the times indicated. Each process will run for the amount of time listed. In answering the questions, use nonpreemptive scheduling, and base all decisions on the information you have at the time the decision must be made.

|  |  |  |
| --- | --- | --- |
| Process | Arrival Time | Burst Time |
| P1 | 0.0 | 8 |
| P2 | 0.4 | 4 |
| P3 | 1.0 | 1 |

A. What is the average turnaround time for these processes with FCFS scheduling algorithm?

**For FCFS scheduling algorithm, the order of execution would be P1, P2, and P3. The completion times for each process would be:**

**P1: 8 P2: 12 P3: 13**

**The turnaround time for each process is the difference between its completion time and arrival time. Therefore:**

**Turnaround time for P1 = 8 - 0 = 8 Turnaround time for P2 = 12 - 0.4 = 11.6 Turnaround time for P3 = 13 - 1 = 12**

**The average turnaround time for these processes with FCFS scheduling algorithm is:**

**(8 + 11.6 + 12) / 3 = 10.2**

B. What is the average turnaround time for these processes with SJF scheduling algorithm?

**For SJF scheduling algorithm, the order of execution would be P1, P2, and P3. Since all processes arrive at different times, there is no need for preemption. The completion times for each process would be:**

**P1: 8 P2: 12 P3: 13**

**The turnaround time for each process is the difference between its completion time and arrival time. Therefore:**

**Turnaround time for P1 = 8 - 0 = 8 Turnaround time for P2 = 12 - 0.4 = 11.6 Turnaround time for P3 = 13 - 1 = 12**

**The average turnaround time for these processes with SJF scheduling algorithm is:**

**(8 + 11.6 + 12) / 3 = 10.2**

9. Consider a virtual address space of 256 pages with a 4 KB page size, mapped onto a physical memory of 64 frames

A. How many bits are required in the virtual address?

**Since the virtual address space consists of 256 pages, it requires log2(256) = 8 bits to represent the page number in the virtual address. Each page size is 4 KB, which requires log2(4 KB) = 12 bits to represent the offset within the page. Therefore, the total number of bits required in the virtual address is 8 + 12 = 20 bits.**

B. How many bits are required in the physical address?

**The physical memory has 64 frames, which requires log2(64) = 6 bits to represent the frame number. Since the page size is 4 KB, which requires log2(4 KB) = 12 bits to represent the offset within the frame, the total number of bits required in the physical address is 6 + 12 = 18 bits.**

10. Consider an operating system that use hardware support for paging to provide virtual memory to applications.

A. Explain how space and time overheads arise from use of paging.

**Paging introduces space and time overheads. The space overhead arises because each process may have its own page table, which means that the same page may exist in multiple page tables. In addition, each page table may require multiple pages to store it, increasing the memory required to manage the page table. The time overhead arises from the need to perform two memory accesses to access a single memory location - one to access the page table, and another to access the page.**

B. Explain how the Translation Lookaside Buffer (TLB) mitigates the time overheads.

**The Translation Lookaside Buffer (TLB) is a hardware cache that stores recently accessed page table entries. When the processor needs to access a page table entry, it first looks in the TLB. If the entry is found, the TLB provides the physical address directly, avoiding the need to access the page table. This greatly reduces the time overhead of paging.**

C. Explain how the multi-level page table mitigate the space overheads.

**Multi-level page tables provide a way to mitigate the space overhead of paging. Instead of a single page table, each process has a multi-level page table. The top level of the page table is a small table that maps virtual pages to a second level page table. The second level page table maps virtual pages to physical pages. By using multiple levels of indirection, the size of each page table is reduced, reducing the space overhead. In addition, because most processes only use a small fraction of their virtual address space, many entries in the page table will be unused. Multi-level page tables can be configured to allocate page tables on demand, so that only those portions of the address space that are actually used require a page table. This further reduces the space overhead of paging.**