Assignment 3

Question 1

System Tracing with Strace

Strace is a system tracing utility that monitors system calls made by processes. Below are detailed explanations of key points in our Strace implementation:

1. **Syscall Creation**:
    - To enable system call tracing, I created a new syscall specifically for Strace.
    - This syscall is designed to provide information about system calls made by processes in the system.
2. **Mask Storage**:
    - Within the `struct proc`, I added an integer variable to store the syscall mask.
    - This mask is used to specify which system calls should be traced for each process.
3. **Parent-to-Child Mask Copy**:
    - In the `proc.c` file, I implemented functionality to copy the syscall mask from a parent process to its child process.
    - This ensures that child processes inherit the same syscall tracing configuration as their parent processes.
4. **Syscall Metadata**:
    - I created an array that holds the names of various system calls along with the corresponding number of arguments for each syscall.
    - This array provides essential metadata for interpreting and logging system calls.
5. **Tracing Execution**:
    - Utilizing the syscall names and argument counts stored in the aforementioned arrays, I implemented the following tracing functionality:
        - I inspect the `trapframe` registers to retrieve the arguments of the executed syscall.
        - I then log these arguments, along with the return value of the syscall.
        - This tracing data can be invaluable for debugging and monitoring the behavior of processes within the operating system.

This Strace implementation enhances the transparency of system activities by providing detailed information about the system calls made by processes, their arguments, and the outcomes of those calls. It serves as a powerful tool for debugging and analysis.

Question 2

First-Come-First-Serve (FCFS) Scheduling Algorithm

The FCFS scheduling algorithm is a simple approach where processes are executed in the order they arrive. Below are detailed explanations of key points in my FCFS implementation:

1. **Process Selection**:
    - In the FCFS algorithm, the scheduler selects the next process to run based on the order in which they were created.
    - I traverse the process table to identify the process that was created first and schedule it to run.
2. **Initialization**:
    - When a process is created using the `allocproc` function, I initialize a specific variable within the `proc` structure with the value of the global `ticks` variable.
    - This timestamp records the time when the process was created, aiding in determining its position in the FCFS queue.
3. **Lock Acquisition**:
    - In the scheduler function, I acquire a lock before evaluating whether the current process is the best candidate to run.
    - This locking mechanism ensures that scheduling decisions are made atomically, preventing concurrent modifications to the scheduling data structures.
    - If a better-suited process is found during evaluation, I release the lock and perform a context switch to that process.
4. **Preemption Disablement**:
    - To prevent preemption, I disable the yield function in `trap.c`.
    - By doing so, I ensure that a process continues to run until it completes its time slice or voluntarily relinquishes control.

Priority-Based Scheduling (PBS) Algorithm

The Priority-Based Scheduling algorithm assigns priorities to processes and schedules them based on these priorities. Below are detailed explanations of key points in my PBS implementation:

1. **Priority Initialization**:
    - Each process created in the system is assigned a static priority of 60.
    - This initial priority is a baseline value used for scheduling decisions.
2. **Tracking Variables**:
    - I maintain various variables to keep track of each process's scheduling statistics, including the number of times a process has been scheduled to run.
3. **Update Time Function**:
    - To monitor and update run and sleep times, I have implemented an `update_time` function.

- This function iterates through the process table, updating the relevant statistics for all processes.

4. **Dynamic Priority Calculation**:
    - My algorithm calculates dynamic priorities based on a predefined formula.
    - The dynamic priority is used to select the process to run based on both its static priority and recent behavior.

Multi-Level Feedback Queue (MLFQ) Scheduling Algorithm

The Multi-Level Feedback Queue (MLFQ) is a dynamic scheduling algorithm that uses multiple queues with different priorities. Below are detailed explanations of key points in my MLFQ implementation:

1. **Queue Framework**:
    - I've introduced a framework for a queue data structure in `proc.h`.
    - Implementation of related functions such as `front`, `push`, and `pop` is done in `proc.c`.

2. **Queue Initialization**:
    - I initialize an array of five queues, each containing elements of the `proc` data structure.
    - These queues represent different priority levels.

3. **Process Queueing**:
    - In the scheduler function, processes that are in the RUNNABLE state but not in any queue are identified and added to the appropriate queue based on their priority.

4. **Time Quantum Enforcement**:
    - I enforce time quantum limits by checking in the `clock_interrupt` function if any process has exceeded its allotted time quantum.

Question 3

Overview

To extend the `procdump` function in XV6, follow these general implementation steps:

1. **Locate the `procdump` Function**:
    - Open the `kernel/proc.c` file in the XV6 source code.

2. **Modify the `procdump` Function**:
    - Inside the `procdump` function, you'll find code responsible for printing information about each process. Modify this code to include the additional information as per the requirements.

3. **Display Additional Information**:
    - For each process, add code to display the specified additional information. You may need to calculate and format this data accordingly.

4. **Formatting**:
    - Ensure that the information is displayed in a well-formatted and readable manner using formatting functions provided by XV6.