Drake Tetreault
ID 35571095

CS 145B – Project 1

Preliminary Document

Introduction

The goal of Project 1 is to simulate a simple operating system that is capable of running a number of processes simultaneously. While running, processes may request access to named system resources or IO devices, both of which kind of request may or may not be possible in the current state of the system. When a process requests access to a resource which is not immediately available, the simulated operating system will block that process without stalling the rest of the system. The system chooses which process to run based on the processes' priorities, where processes with the same priority are granted CPU time based on a round robin scheduling algorithm.

The deliverable requirements are:

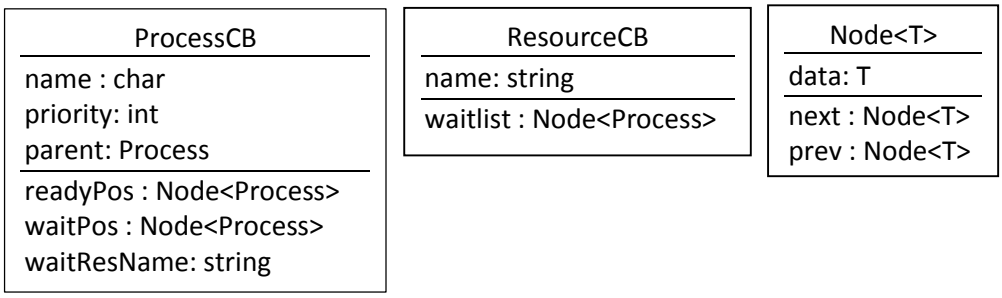
- This preliminary document.
- A revised final document.
- The source code and an executable program.
- A meeting with the class TA to demonstrate functionality.

Data Structures

The first main data structure I will need to implement will store all information regarding a process' status, and will be my system's equivalent of a Process Control Block. In addition to immutable process information such as the process' name, priority, and parent, this data structure will have references to the linked list nodes that represent the process' position in the ready queue and a waiting queue.

The second data structure I will need to implement will store all information about a resource's status, and will be my system's equivalent of a Resource Control Block. This data structure will store both resources name and a linked list of process access requests which will be served in FIFO order.

The last main data structure I will implement are the linked lists that will compose the ready and resource waiting queues. For linear time insertion and deletion, my linked list nodes will maintain pointers to the next and previous elements in the list, as well as a reference to the data object.



Drake Tetreault
ID 35571095

System Architecture

My initial design divides functionality among the following components:

- The entry point. This component is responsible for initializing the system (including parsing command line arguments) and routing input to the command manager so that component can dispatch the command to the process manager or resource manager as appropriate.
- The process manager. This component is responsible for creating and deleting processing and ensuring the entire process tree is deleted when a process with children is deleted. This component will also store the ready list.
- The resource manager. This component is responsible for managing all resources in the system, and responding to process requests for resource access and release. This component will maintain each resource's waiting list.
- The command manager. This component knows about every available command in the system, and also knows about the process manager and resource manager in order to notify them as needed after parsing each input into a command.

Test Cases

To test my implementation, I will first execute the provided sample input and confirm that it matches the expected sample output. I will then write several tests of my own and manually simulate a small system, making sure my program outputs the expected results.

Pseudocode

My program will function according to the following high level overview.

```
do_init():
        create process manager with empty ready list
        create resource manager with all resources free
        create command manager
main():
        do_init()
        quit := false
        while(!quit)
                command := read_line()
                switch(command):
                        "init": do_init()
                        "quit": quit := true
                        "cr": create_process(command)
                        "de": destroy_process(command)
                        ... // Handle all possible commands
create_process(command):
        if duplicate name:
                report error
        else:
                create new PCB
                attach as child of currently running process
                append to end of appropriate ready list
                set status to 'ready'
destroy_process(command):
        if no active process:
                report error
        else if no process with command name:
                report error:
        else if process to destroy not child of active process:
                report error
        else:
                destroy_process_tree(process to destroy)
... // Further command handling functions for all other possible commands
```