Lab 4 Report

Memory Allocation Algorithm:

Let memory represent an array of 1024 values

Let n represent the amount of memory a process requires

int allocateMemory (int n)

for i in range(0..length(memory)-64)

if memory[i] == 0

if i+n > length(memory)-64

return -1

for j in range(i..i+n)

if memory[j] == 0

memory[j] = 1

else

for k in range (j..i), k-- // iterate backwards

memory[k] = 0

return -1

return i

This is a first free algorithm which selects the first available memory location (i.e. a 0) and allocates from there. If it reaches memory that has already been allocated (i.e. a 1) it must free the memory and find the next available location. This method was chosen since it was the easiest. It is by no means the most efficient. Better algorithms could scan the memory first to see if there is a big enough block available.

Structures:

One type of queue used by the process dispatchere is the FIFO, or First In First Out queue. This queue is used for running Real-Time priority processes, as well as 3 more queues for each of the 3 levels of the User-Level priority processes. These queues will be implemented as linked lists.

A structure called “resources” is used to keep track of available resources. It contains integers for the number of printers, the number of scanners, the number of modems and the number of CD drives, as well as an array of integers representing the available and allocated memory. Allocated memory is represented be a block of ones, and free memory is represented by a block of zeroes.

Overall Structure:

The code is divided among four source files.

sigtrap.c was provided and contains the code to display to the console all the signals sent to processes.

queue.c contains all the code for the queue data structure.

hostd.c contains the main method, and the executeQueue method which forks and execs the next process in the queue.

utility.c contains various utility functions, including a function to create a process, and functions to allocate and free the various resources.

Multilevel dispatching:

Multilevel queue scheduling allows for processes to be executed based on the priority of the process. It is useful for systems with foreground and background processes. Our program uses a multilevel feedback queue which allows processes to be demoted to lower level queues. Real operating systems like OS X and Windows use multilevel feedback queues which also allow processes to be promoted to higher level queues.