OS Lab 4 - HOST Dispatcher Shell

1.a) There are several different kinds of memory allocation schemes that could be used, as long as it isn't paging memory or virtual memory. Possible schemes include first fit, next fit, best fit, worst fit and the buddy memory allocation algorithm.

I. **First Fit Algorithm**
   A. The first fit algorithm works by looking at the process queue and taking the first process that is waiting. Then the algorithm searches through all the free memory blocks, starting at the beginning, and as soon as it sees a partition big enough to fit the process it allocates that partition to the process. It doesn't matter how small or large the partition is as long as the process fits. The algorithm then returns to the beginning of the free memory and does the same with the next process in the queue.
   B. Advantages: Relatively fast at allocating memory since it accepts the first possible solution. Able to choose where freed blocks of memory go in the free memory block list.
   C. Disadvantages: Restarts at the beginning of the free memory blocks every time, so memory blocks later in the list might spend long periods of time inactive. As a result first fit wastes a lot of memory.

II. Next Fit Algorithm
   A. A variation of the first fit algorithm, for the first process in the list it performs the same as the first fit algorithm. However when it starts allocating subsequent processes, it starts at the memory position that the previous process was allocated at. This way, the entirety of the free memory array gets traversed more often.
   B. Advantages: Less memory waste than first fit, and still relatively fast at allocation.
   C. Disadvantages: Need to keep track of the previous memory position from the last iteration.

III. Best Fit Algorithm
   A. The best fit algorithm searches the free memory until it finds the optimal block of memory for the process. The optimal memory block wastes as little memory as possible, with the total size of the block being as close to the process memory requirements as possible.
   B. Advantages: Amazing memory management, very little memory is wasted.
   C. Disadvantages: Since it searches for the most optimal memory block, allocation is much slower than first fit and next fit.

IV. Worst Fit Algorithm
   A. Worst fit is the opposite of best fit. Instead of finding the most optimal memory block for the process, it assigns the process to the largest available memory block.
   B. Advantages: Easy to implement.

C. Disadvantages: Huge amount of memory waste, and slow to allocate since algorithm must scan for largest memory block.

V. Buddy Memory Allocation

A. In the buddy memory algorithm, the allocator will partition the memory blocks into blocks of certain sizes(usually powers of 2).The allocator will keep lists of memory blocks of each size. When allocating a process, it rounds the process memory up to the next largest block size, then allocates the process to the first free block of that size. If no such block is free, it takes a block from the next largest size and breaks it into smaller pieces so it can fit the process.

B. Advantages: Can be implemented using a binary tree, which makes traversal very simple and efficient.

C. Disadvantages: Can have bad memory management depending on what size the processes are relative to the block sizes used.

We have decided to use the first fit allocation scheme, since it is easy to implement while still having an efficient allocation speed. It's memory management isn't the best, but since we're dealing with relatively small amounts of memory and not many processes there shouldn't be much of an issue with a less memory efficient algorithm.

b) Describe and discuss the structures used by the dispatcher for queueing, dispatching, and allocating memory and other resources.

- **Queueing**
  - There are four queues in the Dispatcher, each queue uses a **linked list** data structure with pop( ) and push( ) functions to remove and add the the queue respectively.

- **Dispatching**
  - When a job is dispatched from a **linked list** of jobs. Depending on the the priority, the job will be assigned to one of the four queues in the dispatcher. Whenever a job is higher priority, any current job will be suspended and the higher priority job will be executed. The suspended queue will be assigned a lower priority level and requeued in that assigned queue.

- **Allocating Memory and Other Resources**
  - I/O will always require <= 64MB of memory for real time jobs.
  - An **array** will be used to manage the 1024MB memory available to the dispatcher.
  - Each block of memory will be assigned to a process.

- **Other Resources**
  - 2 printers, 1 scanner, 1 modem, 2 CD drives, 1024MB memory
  - These resources will be stored in a **struct**

- Every time resources are used, the value of the resource can be changed/allocated to the process.

c) Describe and justify the overall structure of your program, describing the various modules and major functions (descriptions of the function "interfaces" are expected).

We decided to use the First Fit Allocation Scheme for allocating memory blocks to processes, due to its high speed in allocating partitions to processes. It's true that it does waste a small amount of the memory but our main objective for this project is speed over efficiency.
We are going to use the following functions in our program:

hostd.h

int main(int argc, char *argv[]): Initializes the program.

utility.h

typedef struct { … } resources; The implementation of the memory resources, will contain an int array of size 1024 to emulate memory. Will also contain data members for other resource constraints.
typedef struct { … } proc; The implementation of a process. Contains various process information like priority, runtime, memory size in megabytes, number of I/O devices, its address in the memory and its arrival time.
int alloc_mem(resources res, int size): Searches through the memory array until it reaches a free block of memory. If the memory block is not large enough, continue to the next block until we find one that fits. If none are found then return false.
free_mem(resources res, int index, int size): Once a process has finished its run time, go to the index of the beginning of the process and free all memory being used by the process.
void load_dispatch(char *dispatch_file, node_t *queue): Loads the processes in the dispatch file into the queue.

queue.h

typedef struct node{ … } node_t; The struct that contains the queue implementation, has two data members. One is a process and the other is a pointer to the next node in the queue.
node_t *push(node_t *head, proc process): Takes a process and pushes it onto the end of the queue.
node_t *pop(node_t *head): Removes the first node in the queue and returns it.

d) Discuss why such a multilevel dispatching scheme would be used, comparing it with schemes used by "real" operating systems. Outline shortcomings in such a scheme, suggesting possible improvements. Include the memory and resource allocation schemes in your discussions.

Such a multilevel dispatching scheme would be used because it is very simple and time efficient.(not sure about comparing it with real OS). The main drawbacks in this system is that unlike real Operating Systems, it does not support Virtual Memory, so it loses the ability to swap memory to disk. As well as the fact that it is not a Paged System. So if by any chance our Job Dispatch List is full and new jobs are trying to enter that list it won't be possible to add them until some jobs are done and some space becomes free to add those jobs.