Khanh Cao Quoc Nguyen
311253865

COMP3520 Assignment 2 Report

Memory Allocation Algorithms

The possible memory allocation algorithms that could have been implemented were:

- The first fit algorithm performs a linear search on the memory structure to find the first unallocated block of memory which is large enough to hold the process.
- Next fit is like first fit but the starting point is at the last allocated block's offset.
- Best fit algorithm finds the the smallest block of memory that will fit the process and allocate it to that process. This requires a complete search on the entire list unless the memory structure is ordered by size, therefore this is the worst performer in terms of speed.
- Worst fit algorithm which fits the largest suitable memory block to the process which is clearly out of the picture since it is by far the worst strategy in terms of speed and storage utilisation.

I chose to implement the first fit algorithm as the memory allocation algorithm for my HOST dispatcher for it's simplicity. The first fit algorithm is by far the easiest to implement which allowed to complete this assignment under the time constraint and with my current programming ability. I also believe due to the nature and size expected of this dispatcher, such naive methods are permitted and would be faster than best fit and worst fit algorithms. The next fit algorithm would have required an extra step to store the current position of the last allocated block of memory and followed by the a linear search from that point, which is essentially the same as the first fit algorithm but from a different starting point.

S**tructures used by the dispatcher**

- Process control block (PCB)

The process control block or PCB is the process structure used by the dispatcher. The structure contains variables such as id, arrival time, remaining CPU time, priority, memory and resource requirements and it's current status. The PCB structure is one of a doubly linked list which allows queuing of processes which is an essential part of the dispatcher.

- Memory structure (MAB)

The memory structure or MAB is used to allocate memory for processes in the dispatcher. The structure contains variables such as offset, size, allocated, process. MAB is also a doubly linked list when the initial block, size of 960, is split up into smaller blocks.

- Resource manager (RSRC)

The resource manager checks, allocates and frees resources that processes require to run. In this system, we have 2 printers, 1 scanner, 1 modem and 2 CDs which are simply global variables initialized on execution of the program.

Khanh Cao Quoc Nguyen
311253865

P**rogram structure and individual modules**

For simplicity and modularity, the dispatcher program was split up into different parts; process structure, memory structure, resource management and a utility helper. These modules are all implemented as header files.

The utility helper only consists of a function that reads each line of an input file and put the job parameters into process structures (PCB) and create a list of processes.

The process structure is implemented as a doubly linked list which allows the dispatcher to queue jobs which is an essential part of this assignment.
This structure has the following functions:

- create new process
- start process
- terminate process
- enqueue process
- dequeue process
- suspend process
- restart process

During the reading of the inputs, the dispatcher would create new processes and store the parameters inputted and queue up the processes with the enqueue function. When the process is ready, it would then call the dequeue function and the start process function which fork and execs the process. If the processes needs to be suspended since it's in round robin, then it would call the suspend and restart functions accordingly.

The memory allocation structure is also a doubly linked list.
It has the following functions:

- create new memory block
- check memory for allocation
- allocate memory block
- free memory block
- split memory block
- merge memory blocks

Memory allocation of this dispatcher uses the first fit algorithm as mentioned previously. On executions, the dispatcher will call the create new memory block function. After the processes are taken from the input queue, it then needs to be allocated memory, so it will call the check memory function which would search through the list to find the first block that can cater for the process requirements. It would split the memory block and form a linked list if needed. When the process is completed, the free memory function is called and it would merge any adjacent blocks that are unallocated.

The resource manager manages the global variables which represent the resources of the system. After memory is allocated, the resource check function is called to check if there are enough resources available for a process, if there is then it would continue to call the resource

allocation function which decrements the global variables accordingly to it's needs. When the process is completed, the free resource allocation is called which increment the respective resource variables.
dispatcher

Dispatching scheme (including memory and resource allocation), shortcomings, and possible improvements

The Hypothetical Operating System Testbed (HOST) is a multiprogramming system with a four level priority process dispatcher operating within the constraints of finite available resources.

On execution of the dispatcher, the read input function is called to obtain the list of jobs from the input file, create process structures (PCB) for each process and queue them up. That queue is then sent to the dispatching function which on execution initialises the user queue, real time queue, the three priority queues and a memory block of 1024MB but since 64MB is reserved for real time processes, the size is only 960MB.

After the input queue is ready, the dispatcher initialises the dispatcher timer which increments every second, if the process in the input arrives then it's priority will be checked. If the priority is 0 then it is a real time process and is sent to the real time queue which is essentially a first come first serve queue and is given priority over the other queues.

On the other hand, if the priority is 1, 2 or 3 then it will go into the user queue to await memory and resource allocation. This is where the first fit memory algorithm comes into play. The process will stay in the queue until it has been allocated both the required memory and resources. Once this happens, it is then put into the appropriate priority queue where the first two queues are in round robin and the third is a feedback queue.

When the queue management part is completed then the dispatcher will start to start processes that are in either the real time queue or the priority queue. The real time queue will take priority and the real time process will run until it is completed (non-preemptive) and then it is terminated.

If there are no real time processes queued up then the dispatcher will dispatch the first job it finds in the priority queues starting at the first one. The processes that are dispatched from the priority queues only run for 1 second and then it is suspended and then put into the next lower priority queue, awaiting for it's turn to be restarted. If it already in the third priority queue, then it just gets fed back into the queue and put at the end.

The timer is then incremented and the queue management and dispatching of processes will continue until there no more jobs left in any of the queues.

Khanh Cao Quoc Nguyen
311253865

Shortcomings of this dispatcher:

- There is possible starvation for user priority processes which may find it hard to get memory and resources or even get at turn to be executed since the real time processes will always have higher priority. An improvement for this would be to set some condition that would raise the priority of the process if it's been in waiting for too long.
- Since the first fit algorithm was implemented there may be some fragmentation at the start of the memory structure, and therefore the next fit algorithm would have been more effective.