HOST DISPATCHER DOCUMENTATION

Author: Ashwin Ramesh
Student ID: 311254012

1. Introduction

The seconds assignment in the Operating System Internals course focuses on the implementation of a hypothetical host dispatcher system. The main inspiration is to try and mimic the actual process dispatching mechanism used by the various operating systems in today's computers.

This design document will, in detail, explain the reasoning behind the memory allocation algorithm used, the various data structures created to make the dispatching algorithm work and possible issues stemming from this implementation and potential improvements in the next iteration of the program.

2. Memory Allocation Algorithms

The specification for the Host Dispatcher Assignment was to ensure that the memory allocation algorithm used was implemented using contiguous allocation. This means that memory is consecutively allocated to each process. The size of the process is compared to the amount of contiguous main memory remaining and if there is a block of memory, the process is placed in memory otherwise it is placed in queue awaiting memory. [1] This constraint resulted in a subset of memory allocation algorithms that could be implemented for this assignment.

2.a. First Fit

The First Fit Algorithm allocates the first large enough contiguous block of memory found to the process. The algorithm uses a linked list type data structure, where each block of memory is connect to its neighbours. It starts at the first block, iterating through each of the next blocks until it finds a large enough block of memory. If that block of memory is too large, it will be split into two separate blocks such that the extra memory portion is left free for other processes. [2]

The obvious advantages of the First Fit algorithm is that it allocates memory very quickly and manages to merge smaller, neighbouring block to become a larger block. However, as the algorithm always starts at the front of the linked list, smaller holes tend to accumulate at the beginning of memory, increasingly fragmenting the whole memory block. [3] This is also known as internal fragmentation.

2.b. Next Fit

As mentioned above, the First Fit algorithm has the downfall of heavy fragmentation at the beginning of the memory block. An improvement would be to leave the pointer to the last allocated location rather than moving back to the start of the list. This would better distribute the allocation process. This method of memory allocation is known as the Next Fit algorithm. However, the Next Fit algorithm does create the issue of external fragmentation, such that the memory allocation causes many small blocks of free memory to be scattered throughout the total memory space. This results in larger processes not being able to run due to a lack of contiguous memory. [4]

2.c. Best Fit

The Best Fit algorithm is much slower than both the First Fit and Next Fit algorithms. Instead of allocating to the first largest block of free contiguous memory, it will search through the whole memory space to find the smallest block which will fit the process memory required. [5] This means that the memory is efficiently allocated. The performance of this algorithm is the worst among the various algorithms described in this report.

2.d. Worst Fit

The Worst Fit algorithm will allocate the largest remaining block to the next process, which will allow future processes to use the remaining memory from that block. The idea is to maximise the split to increase the probability of allocating memory to other processes. [6] This reduces the chances of external fragmentation but is also very slow to run.

2.e. Buddy System

Unlike the various "Fit" systems described above, which use a linear linked list data structure to store contiguous memory blocks, the buddy system uses a binary tree type data structure to allocate memory between processes. The allocator will only assign blocks of specific sizes, and has many free lists one for each permitted size. Usually these sizes are powers of two.

If the requested memory size is not exactly available, then the algorithm will split a larger block in two again and again until a block of the correct size is created. The main advantage of this system is that merging (coalescence) of adjacent blocks is cheap since the neighbour of any free block can be calculated from its address.

This type of system can either react well to incoming processes or can cause severe internal fragmentation due to the rounding of block sizes. Typically, the process of freeing and merging of memory is extremely fast, maxing out at $\log\left(\frac{\text{max. block size}}{\text{min. block size}}\right)$. [7]

2.f. Personal Choice

For this assignment, my personal choice was to use the First Fit Algorithm. Although the Buddy System does offer a better memory management scheme in the long run, it is far too complex for this type of assignment. On the other hand, the other “Fit” algorithms were not as efficient as the First Fit or resulted in a far more fragmented memory allocation. Finally, the First Fit algorithm is extremely easy to implement, allowing me to focus on the actual dispatcher and queuing system in more detail.

3. Data Structures used in the Host Dispatcher

This section of the report will outline the various data structures and algorithms used to create this hypothetical host dispatcher.

3.a. Process Structure

The process data structure is fairly simplistic. It stores the process id created by the system, various I/O devices, memory size required as well as its arrival time and processing time. The processes also work in a linked list nature, allowing easy en-queuing and de-queuing from the various queues. All memory allocated to this process will be retained for the life of the process, once the processing time is up, the memory and the process will be freed.

3.a. Queue Structure

The Host Dispatcher is implemented using six different queues, each utilising an underlying linked list. As illustrated on the diagram below, there is a multi-level feedback queue such that any process having completed one iteration of the clock will have it's priority reduced (unless it is already on the lowest priority level). For example, a process beginning on priority 1 will move to priority 2 then priority 3 over the course of it's processing.

For this assignment, the maximum amount of processes is 1000, which will be read from an input file, into the Job Dispatch List. From here, items are allocated to either the Real Time or User Job queues when the arrival time of the process matches the current system time.

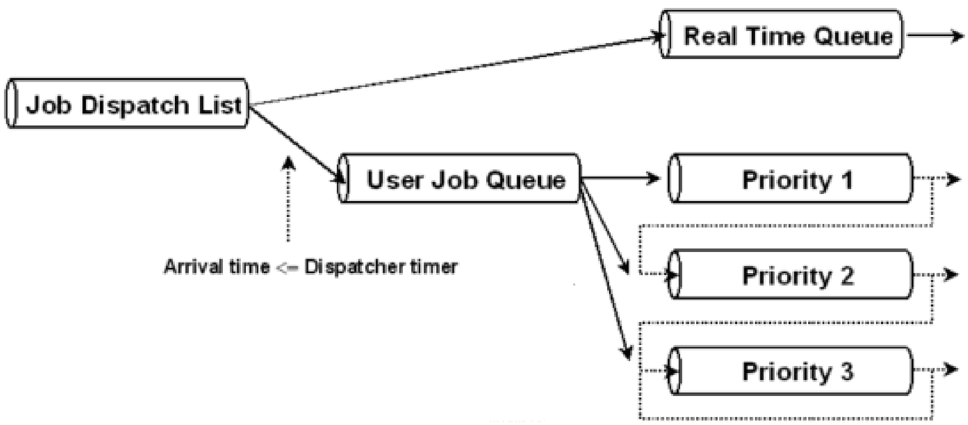

Figure 1

The feedback queues only run for “user job” processes, or those processes that require I/O resources. They run on priority levels one through to three. On the other hand, “real-time” processes are automatically priority zero, allowing them to continually run before any user jobs.

3.b. Memory Structure

The memory allocation block (MAB) is also implemented using a linked list data structure. From the assignment specification, all real time processes will automatically be allocated 64 MB of memory, while user jobs will be allocated 960 MB altogether. This adds up to 1024 MB of memory.

As described in section 2 of this report, the First Fit algorithm will be used, meaning that after finding a suitable MAB, any extra MB will be split into a separate MAB. Similarly, when the memory is freed, the MAB will try to merge with it's neighbours to create a larger block. All user jobs will only be queued into the feedback queues when an appropriate MAB is located.

3.c. Resource Structure

I/O resource allocation uses a simple data structure storing the maximum I/O devices available. In this system, there are 2 printers, 1 scanner, 1 modem and 2 CDs. This can easily be changed by re declaring values in the source code. Similar to memory, user jobs cannot enter the feedback queues until resources are freed. Each process carries its resources until the process is terminated.

3.d. Dispatcher Structure

With the use of the aforementioned structures, the dispatcher runs while there are still any processes remaining to run or while a process is currently running. The dispatcher's main role is to place processes onto correct queues and simultaneously allocate memory and resources using the systems mentioned previously. As a hypothetical system, the dispatcher does not actually process using I/O devices or actual processes, but rather uses a predefined program provided by the University.

Program Structure and Modules

Issues with the Dispatching Scheme and Possible Improvements

References

1. http://www.techopedia.com/definition/3769/contiguous-memory-allocation
2. http://www.memorymanagement.org/articles/alloc.html
3. http://www2.cs.uregina.ca/~hamilton/courses/330/notes/memory/segmentation.html
4. http://wiki.answers.com/Q/What_is_the_difference_between_external_and_internal_fragmentation
5. http://courses.cs.vt.edu/csonline/OS/Lessons/MemoryAllocation/
6. http://research.cs.vt.edu/AVresearch/MMtutorial/WorstFit.php
7. http://www.memorymanagement.org/articles/alloc.html
8.