Name: Victor Woo
SID - 430323825

COMP3520

Design Documentation

1. Describe and discuss what memory allocation algorithms you could have used and justify your final design choice.

Dynamic Partition Allocation was used in the host dispatcher since it can partition the main memory dynamically to the exact size of the process's required memory and eliminate internal fragmentation problems. The allocation algorithms schemes which prescribed were First Fit Algorithm, Best Fit Algorithm, Worst Fit Algorithm, Next Fit Algorithm and Buddy System Algorithm.

'First Fit Algorithm' searches through each free memory spaces in the memory block until it finds a memory space which is large enough for the process to be allocated. The advantages of this method is that it is efficient in terms of process allocation since it finds the first memory space which is equal or more size for the process to run in. However internal fragmentation will occur for processes who have long runtime or many processes with varying sizes since it is only finding the first available block meaning there is a chance that there will be unused blocks when the process has been allocated and deallocated.

'Best Fit Algorithm' searches through every free memory spaces in the memory block to find a memory space which is unallocated and equal or similar size. This algorithm maximises the memory block's efficiency in terms of memory utilisation since it finds the best memory space which is equal or similar size for the process to run in. However the algorithm is inefficient in terms of search speeds through the whole memory block, since it has to check all free memory spaces in the memory block. Also internal fragmentation may occur if it places a long running process which does not take up all of the free memory space allocated. If the free space remaining is not used for a long time, it will reduce the overall performance and utilisation of memory.

'Worst Fit Algorithm' searches through every free memory space in the memory block to find a memory space which has the largest free block. The advantages of this algorithm is reducing external fragmentation since it utilises all of its available memory space. However the disadvantages is that the algorithm will have a large internal fragmentation depending on the size of the process to the size of the memory allocated. This will create a very inefficient dispatcher in terms of searching speeds and memory utilisation.

'Next Fit Algorithm' is very similar to 'First Fit Algorithm' however you start searching at the offset of where you allocated your previous process. This method helps speeds up the search to find a memory

space which is large enough for the process can be allocated. However empirically, studies have shown that first fit is more efficient.

'Buddy System Algorithm' combines dynamic partition allocation and fixed partition allocation. Fixed partition allocation is when main memory is partitioned into fixed sized memory blocks. The number of memory blocks depends on the amount of jobs available. Buddy System creates fixed sized memory blocks of different sizes usually through powers of 2. Processes are allocated to their closest free rounded up memory size block this allows the memory block to be used efficiently in allocating and deallocating. If no block are available then it will split the highest memory sized block to fit the process. The advantages of buddy system is that there is little external fragmentation and utilises compaction with little cost to CPU time. Also the allocation and deallocation speeds of processes are fast. However due to allocating fixed block sizes to a varying process memory size, internal fragmentation will occur since a larger block will be requested but the process may not utilise all of the space.

'Paging' is a method which split memory into blocks having the same size. These blocks are called frames and their sizes are usually a power of 2. When processes arrive, they are divided and stored inside the frames. When the process is executed, the pages are loaded into the the frames, however they do not have to be in contiguous order. The OS keeps track of each free and used pages by using a page table. The advantages are elimination of external fragmentation. This is because the processes can loaded non-contiguously to the frames. This allows the process to store in memory in any available memory. Also the allocation of memory to processes is very easy due to non-contiguous memory nature. The disadvantages are internal fragmentation may occur due to process not using all of its memory in the given frames and longer access times since each process has be searched on the page table and size of page table may be very large depending on the amount of processes running.

'Segmentation' is an alternative method to paging where blocks of memory are split into varied sizes which are called segments however there is a set maximum size. The addresses consists of an offset and a segment number. The segment is the index in the page or segment table and the offset is the offset from the beginning of the segment which identifies the elements which is needed.

The allocation scheme which was used in my dispatcher was the 'First Fit Algorithm'. The algorithm is easy to implement and has a fast search time since it finds the first memory space which is equal or more size for the process to run in. For this host dispatcher assignment, since the size of the dispatcher is 1024, it is relatively quick to find an unallocated space for the process. The memory utilisation for the algorithm is excellent since external fragmentation is less likely to occur since there are only small numbers of processes which will be executed through the dispatcher. The Buddy system was also considered due to the efficient memory utilisation (Better than first fit), however the difficulty in implementing the algorithm was one of the factors of not being written in the assignment.

2. Describe and discuss the structures used by the dispatcher for queuing, dispatching and allocating memory and other resources.

Several Queues were implemented to organise the input processes being inserted into the dispatcher. This was necessary since queues manage their data through by using the First Come First Served (FCFS) algorithm. This allowed the process time to be allocated fairly and have a suitable turnaround time. The host dispatcher utilised many queues to enforce its scheduling criteria and fairness to all processes; A real time and 3 priority queues (High, Medium, Low) were implemented to optimise the Turnaround time, Response time, Waiting time, CPU utilisation and Throughput. Each quantum or 1 second the process was suspended and the priority of the process was decreased until it was in the lowest priority queue.

Doubly Linked Lists were implemented to organise the processes and memory allocation. Doubly Linked Lists consists of a group of nodes that forms a list or sequence. The list has two references, one to the next node and one to the previous node which allows you traverse back or forth between each nodes. This allowed the dispatcher to utilise First Come First Serve Algorithm allowing it to traverse the memory space to find an unallocated block to insert the process. Without the linked list, finding a free memory block would be more difficult. Using linked lists for processes allowed enqueuing and dequeuing processes off the queue easier as it was easier to locate the head and tail of the queue. The real time queue only allowed to execute 1 job. However if the job wanted to allocate more memory than the given set memory block (64MB) or ask for any resources then an error and would be deleted from the input queue. A user job queue was implemented to store the processes which were inserted through the job dispatcher list according to their arrival time utilising FCFS. The user job queue contained 960MB of memory and the processes were allocated to specific priority queues by using the First Come First Served Algorithm. If block which the process allocated was bigger than the required process memory size, then it would be split into two blocks; One which was the process which was allocated and the other which was the leftover. When the process cpu time was over, it was merged back into the memory block.

Structs were implemented to organise processes, memory allocation and resource allocation. Structs were important part of the dispatcher since each specific part of the dispatcher needed group of variables to define the structure. Processes needed to define the arrival time, priority, cpu time, memory which needs to be allocated and a pointer to both resources and memory. The memory needed for the process is defined by the size needed to be allocated and the offset from the memory block. The resource structure defined the resources available for the host dispatcher throughout its running time. These resources were 2 Printers, 1 Scanner, 1 modem and 2 CD's.

3.Description and justification of the program structure and individual modules

The 'Host Dispatcher' is a multiprogramming system which connected all of the modules and interfaces together. The hostd.c interface contained checkRSRC() and a main function. The checkRSRC checked if realtime processes were asking for any resources or attempting to allocate more than the defined real time memory size (64MB) and user processes which were asking for more resources than the maximum unused resources or attempted to allocate more than the defined user job memory size (960MB). The main function contained many functions including reading the input from the file specified in the command line and the dequeueing of a process in the input queue. It uses the checkRSRC() function to check the validity of the queue and if valid it enqueues the process to the appropriate queue (Realtime or User Job). Dequing a process from the user job queue was another function where it checked if the memory block had enough space for the process & the resource block has enough space for the process to allocate memory and resources to the process. After it enqueued it back to the appropriate queue based off the process's priority.

The 'Memory Allocation Block' contained functions which help control the allocation, freeing, checking, splitting and merging of memory. memchk() is a function which utilises the First Come First Serve Algorithm to find the first free memory block which can allocate the process memory size. memAlloc() allocates memory to the memory space found from memchk(). If the memory block is larger than the process's memory size then it will split the block where the remaining leftover block will be unallocated and unused. memFree() frees the memory process block given by the process and merge it back to memory block if there are blocks previous or next to it. memMerge() merges the between the given block any or both adjacent block if there are any present. memSplit() splits the process into two memory spaces only if the memory given is smaller than the memory allocated. memPrint() prints out the content of the process's memory offset, size and if it's allocated.

The 'Process Control Block' contained functions which help start, suspend, terminate, enqueue and dequeue processes. startPcb() essentially starts or resumes a suspended process. suspendPcb() suspends the process by calling the SIGSTP signal. terminatePcb() terminates the process by calling the terminatePcb() signal. printPcb() prints out the PID, process memory blocks and resources allocations attributes while printPcbHdr() prints out the titles for each of the Pcb values. createnullPcb() creates and initialises an inactive Pcb. enqPcb() enqueues the given process to the end of the queue whilst deqPcb() dequeues the given process from the start of the queue.

The 'Resource Control Block' controls resources allocations for all processes. rsrcChk() checks if the resources can be allocated for the process. Whilst rsrcChkMax() checks if there could be resources available for the process. rsrcAlloc() allocates I/O resources when the process is first initiated while rsrcFree() deallocated I/O resources when the process has been completed and terminated.

The justification for using this program structure is the design and implementation allowed each of the modules to blend together. All of the modules checked its own validation for example, the resources checked

that no processes could allocate than the maximum available resources, memory checked if processes did not ask for more memory spaces than there was available. If these modules were not split up to multiple programs and sub functions then the readability and usability would be difficult.

4. 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.

The Hypothetical Operating System Testbed (HOST) is a multileveled feedback queue which gives preferences I/O bound processes and shorter jobs which are prioritised in terms on how long the process has ran inside the dispatcher. The highest priority, real time queue is a non-preemptive scheduling queue which executes the process until the process has finished. These processes in the real time queue only have a maximum memory size of 64MB and cannot be allocated any resources. If there are no processes in the real time queue, processes are placed into the user job queue. Every quantum (for this assignment it is 1 second) the current process's priority will be decremented to a lower priority round robin queue and the process is taken out of there are higher priority processes waiting in the queue. This helps the fairness for all processes with long runtime jobs while favouring short run time jobs.
Early Windows OS used non-preemptive scheduling which is similar to just the real time queue aspect of our host dispatcher. However the major difference is that Windows used 32 priority queues in comparison to only 4 in our dispatcher. These queues have 16 different priorities between each real time and priority queue. Similarly to the host dispatcher, the processes's priority inside the real time queues in Windows do not change. However inside these processes are run as a round robin queue allowing different processes to be run inside the queue. Another difference is how Window's sets priorities to processes. The processes initiated as a variable priority class determined by a base priority and thread base priority. Similarly to the Host Dispatcher, if the process's priority changes, it will move to a different queue based off of his priority. The thread base priority may fluctuate in priority, for example if a process is waiting for an I/O process. The kernel however if its interrupted due to the process using up all of its quantum it will decrease the priority. This shows that Windows favours I/O bound processes in comparison to process-bound.
While Linux used a multilevel feedback queue with 140 queues where real time queues take 0 - 99 priority and normal user jobs take 100 - 140. Similar to the Host Dispatcher and Windows, the real time queue takes precedence over normal priority queues. The quantum time for real time processes in Linux is 200ms while user jobs are 10ms which is much faster than the host dispatcher which was 1 second.

This scheme allows short running processes to have a small turnaround time while longer processes will have an equal chance of finishing if the process reaches to the lowest priority queue. This is due to queues being a

round robin queue where they have an equal chance of being chosen as the next process if there are no other high priority processes. Processes which are placed in the real time queue will be non-preemptive ensuring deadlines for high priority processes will be met. In terms of memory allocation a different algorithm could have been implemented instead of First Come First Serve, such as the Buddy System. The Buddy System has a better memory utilisation than FCFS. This could further help reduce internal fragmentation thus in turn increasing the overall performance and search time of the dispatcher. In terms of resource allocation, the dispatcher was very limited to amount of the resources it could allocate. These resources were not realistic in comparison to Linux and Windows schedulers and would cause deadlocks and or starvation to occur. If there was an increase of resources a load balancing policy should be implemented so a process do not use all of the resources reducing the chances of errors due to deadlocks and starvation to occur.

Website Links

http://www.go4expert.com/articles/memory-allocation-schemes-t22406/

http://en.wikipedia.org/wiki/Memory_management_(operating_systems)

http://sparth.u-aizu.ac.jp/CLASSES/OS/LECTURES/lec6/img22.html

http://www.scribd.com/doc/59021292/Types-of-Memory-Allocation-Schemes#scribd

http://en.wikipedia.org/wiki/Fragmentation_%28computing%29

http://en.wikipedia.org/wiki/Paging

http://www.tutorialspoint.com/operating_system/os_memory_management.htm

https://www.it.uu.se/edu/course/homepage/oskomp/vt07/lectures/paging/handout.pdf

www.sunyit.edu/~sengupta/CSC521/Segmentation11.ppt

https://elearning.sydney.edu.au/bbcswebdav/pid-2850726-dt-content-rid-14531250_1/courses/2015_S1C_COMP3520_ND/lectures/Chapter09-15.pdf

http://en.wikipedia.org/wiki/Scheduling_(computing)#Short-term_scheduling

http://www.cim.mcgill.ca/~franco/OpSys-304-427/lecture-notes/node39.html

http://www.answers.com/Q/What_is_worst_fit_algorithm

http://www.memorymanagement.org/mmref/alloc.html

http://en.wikipedia.org/wiki/Buddy_memory_allocation

http://en.wikipedia.org/wiki/Linked_list

http://en.wikipedia.org/wiki/Struct_%28C_programming_language%29

http://en.wikipedia.org/wiki/Multilevel_queue

http://en.wikipedia.org/wiki/Scheduling_%28computing%29

http://en.wikipedia.org/wiki/Multilevel_feedback_queue

http://criticalblue.com/news/wp-content/uploads/2013/12/linux_scheduler_notes_final.pdf

https://www.cs.columbia.edu/~smb/classes/s06-4118/l13.pdf

Operating Systems Internals and Design Principle William - Stallings 2012