**CG2271 Real Time Operating Systems**

**2013/14 Semester I**

**Term Assignment Report**

|  |  |
| --- | --- |
| Matric No: Prachi Singhal | Name: A0103527N |
| Matric No: Vu Ngoc Quang | Name: A0099878W |

Section 1 – Implementation

**Cut and paste your implementation for first fit here:**

// Search for free memory using first fit policy

TMemoryNode\* current = \_head;

while(current != NULL){

if(current->len >= requestedLen && !current->allocated)

return current;

else

current = current->next;

}

// in case memory is not found, return NULL

return current;

**Briefly explain your implementation:**

First fit implementation starts the search from the head and finds the first memory node that is empty and has memory available that is greater than or equal to the requested amount of memory. The current pointer traverses through the linked list and returns the pointer to the first memory node that has enough memory to allocate the requested amount.

**Cut and paste your implementation for best fit here:**

// Search for free memory using best fit policy

TMemoryNode\* current = \_head;

TMemoryNode\* current\_min = NULL;

int i = 1;

while(current != NULL){

if(current->len >= requestedLen && !current->allocated){

if(i==1){

current\_min = current; //this will happen only once

i++; //when current starts traversing

}else if(current->len < current\_min->len){

current\_min = current;

}

}

current = current->next;

}

return current\_min;

**Briefly explain your implementation:**

Best fit will traverse through the linked list and find the memory node that has the smallest available memory and which is sufficient to allocate the requested amount. Firstly, only non-allocated nodes with size is equal or greater than requestedLen are checked. When first traversing, assign current\_min to wherever the current pointer is. If another node is found with smaller size than current\_min, then current\_min will be updated to that node. Eventually, current\_min is returned. In the case that no node is acceptable, current\_min will be NULL, which is then returned.

**Cut and paste your implementation for worst fit here:**

// Search for free memory worst fit policy

TMemoryNode \*current = \_head;

TMemoryNode \*current\_max = \_head; //store the location with max length

//find the largest free memory block

while(current != NULL) {

if (!current->allocated && current->len > current\_max->len) {

current\_max = current;

}

current = current->next;

}

//check if max is fit for requestedLen

if (current\_max != NULL && current\_max->len >= requestedLen) {

return current\_max;

}

else {

return NULL;

}

**Briefly explain your implementation:**

Worst fit searches for the largest non-allocated memory block that can fit the requested size. In our implementation, current\_max pointer is the pointer to the maximum available memory node. Firstly, current pointer traverses the list to find the maximum available memory to assign to current\_max. Then the code will check if the max is fit for requested size, if yes, returns it or returns NULL otherwise.

SECTION 2 – EXPERIMENT 1 : RUNNING TIMES

1. Record your readings below:

Running times:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Algo | Run1  Time | Run2  Time | Run3  Time | Run4  Time | Run5  Time | Average |
| First  Fit | **0.78571429** | **0.78571429** | **0.85714286** | **0.71428571** | **0.85714286** | 0.8 |
| Best  Fit | **0.64285714** | **0.78571429** | **0.85714286** | **0.78571429** | **0.78571429** | 0.771428574 |
| Worst  Fit | **0.78571429** | **0.78571429** | **0.85714286** | **0.78571429** | **0.78571429** | 0.8 |

Variance:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Policy | Run1  Variance | Run2  Variance | Run3  Variance | Run4  Variance | Run5  Variance | Average |
| First  Fit | **0.88461538** | **0.90659341** | **0.69230769** | **0.79120879** | **0.56043956** | 0.767032966 |
| Best  Fit | **0.66483516** | **0.42307692** | **0.45054945** | **0.77472527** | **0.77472527** | 0.617582414 |
| Worst  Fit | **0.51098901** | **0.57692308** | **0.79120879** | **0.77472527** | **0.68681319** | 0.668131868 |

**2. Is there an algorithm that is faster than the others? If there is an algorithm that is faster explain why. If not, explain why as well.**

From the values in the table, it can be inferred that for the tests run in this case, the three algorithm’s performance doesn’t vary greatly. In terms of timing, Best Fit is faster than First Fit and Worst Fit by a very small margin of (0.8-0.7714)ms. Theoretically, First Fit is likely to be faster as the memory is allocated as soon as a possible match is found and there is no extra time spend searching for the largest or the smallest fitting memory. If we expand the scope of the tests, perhaps the timing will vary more for the three algorithms.

SECTION 3 – EXPERIMENT 2: EXTERNAL FRAGMENTATION

**3. External fragmentation is given by:**

**If this formula returns a value of 0.3, what is the significance of this number with regards to memory allocation?**

The ratio LargestFreeBlock/TotalFreeMemory will be 1-0.3 = 0.7, which means that largest free block occupies a large portion of the memory. The memory does not contain many small chunks of memory that cannot fit any requests. Hence, the smaller the value of the given formula, the less externally fragmented the memory is and the more efficient the memory allocation is.

**4. Record the readings from Experiment 2 below:**

|  |  |  |  |
| --- | --- | --- | --- |
| Policy | Total Free Memory | Largest Free Memory Segment | External Fragmentation |
| First Fit | 20.00 | 8.00 | 0.60 |
| Best Fit | 20.00 | 8.00 | 0.60 |
| Worst Fit | 20.00 | 8.00 | 0.60 |

**5. Based on your readings above comment on the effectiveness of each policy at minimizing external fragmentation. Explain your answer based on what you know of each policy.**

Theoretically, worst fit should give the lowest external defragmentation value as it allocates the largest available block to a request. Best fit will give the highest external defragmentation because it will leave many small free blocks which can hardly fit future requests.

However, based on the experiment results, all three allocation algorithms give the same external defragmentation because there are too few tests run in this case. If the scope of the tests run is increased, the fragmentation values will most likely differ.