April 21, 2016

Nathan Lea

CS4323

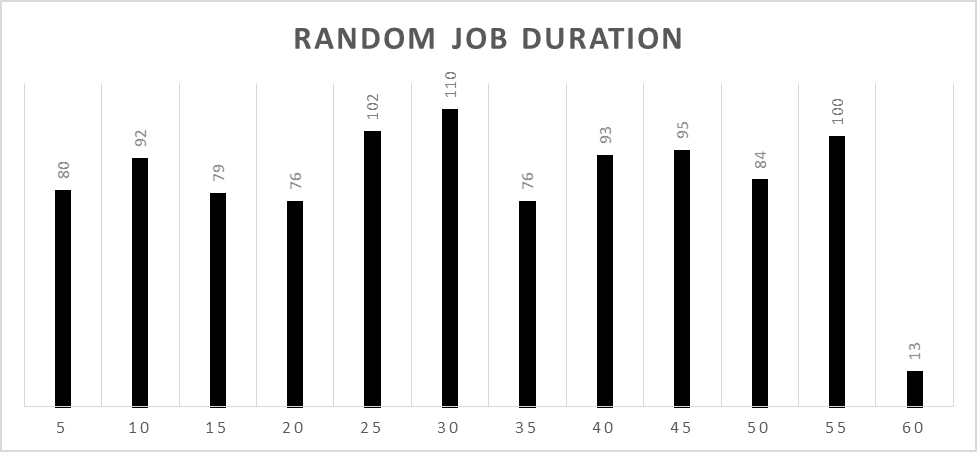
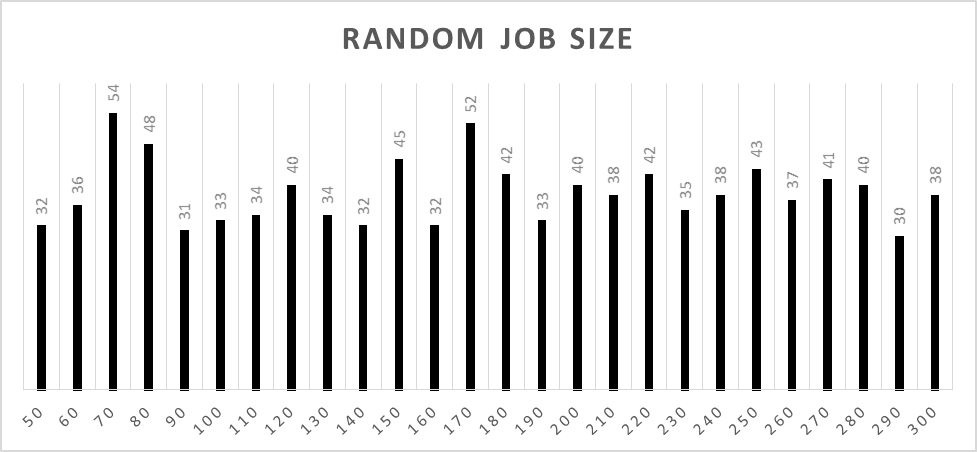
Operating Systems  
memory simulation

Project – Phase 2

# Random Number Generator:

The random number generator I used in my project is the SecureRandom Object. This random number is a cryptographically strong random number generator which produces a nondeterministic output. This ensures that the jobs that are being produced by the simulation are truly random and cannot be guessed based on the last job.

Below is an output of the secure random number generator scaled to the correct mappings for job size and duration respectively each was over 1000 runs:



This graphs show that the random number generator is not symmetric and does not have a normalized distribution, which means that it is truly a random number generator.

# Adaptions from Phase 1

Much of the project is very similar to phase 1, but there are several pieces that have been changed to fit the new guidelines for phase 2. The central idea of the simulation has not changed, job is initialized, job is attempted to be placed in memory according to a certain strategy, job is executed, and finally the job is removed from memory. The method for doing this has been altered which forced several changes in several areas in the simulation such as the simulation approach, job representation, memory manager, the addition of the pending list and the round robin scheduler, and coalescence and compaction has been added.

## Simulation Approach

The simulation is event based with a 5 VTU increase each time through the simulation loop because of the desired Round Robin Schedule with a time quantum of 5.

## Job Representation

A job is represented by a block of memory that is equivalent to the size of that job. For easy of locating and gathering data about the job, each job has a job header represented below:

|  |  |
| --- | --- |
| PID | Process Identifier |
| DURATION REMAINING | Duration left of the Job |
| DURATION | Duration of the Job |
| VTU | Time job entered memory |
| PID | Process Identifier |

Every job in memory will have this header, if the job is larger than 50K then the rest of memory occupied by the job is filled with the PID of the job. Job PIDs are implemented in a similar way to a basic UNIX system where as each job is created the PID is increased by one.

Duration remaining is the time that is left to execute for the job, each time the CPU is allowed to execute on the job this number is decreased by the time is was executed. This allows the CPU to know how much time is left on the job and when to kick the job because the job has completed.

## Memory Manager

The memory manager is given a job and based on what algorithm the simulation is using, first, best, or worst fit, it will attempt to place the job in memory. If there a hole that is able to fit the job it adds the job PID to the ready queue. If it is unable to find a hole large enough in memory it places the job headed into the pending list. It will not ever reject job since there will always be a large enough hole at some point in the simulation to fit the job.

## Pending List

The pending list is an array of job header of which were unable to fit into memory when they first arrived in the simulation. This pending list is checking top to bottom every time a job is completed to see if it can fit any jobs in memory. This is truly a list of jobs and thus the entire list is checked and it will attempt to fit all jobs in memory every time a job completes. As per requirement this list has priority over any new jobs arriving in the system.

This method of FIFO pending list has a problem where the simulation is given priority to shorter jobs and as the simulation rolls on there is mostly likely starvation for large jobs in the pending list. This however will allow for the simulation to complete more jobs since the simulation is not being stalled by waiting on memory for a large hole to open up. With the addition of compaction and coalescence this stalling for large jobs is even less of a problem, since all the small holes will create one large hole.

## Round Robin Scheduler

The ready queue is a circular queue that contains all the jobs currently in memory. As a job is executed on is moved to the bottom of the list.

### CPU

The CPU has a time quantum of 5 VTU, which means is only operates for 5 VTUs per job per loop through the simulation, each time a job is operated on the durations remaining header is reduced by 5.

### Dispatcher

The dispatcher is responsible for removing the job from memory when the remaining duration is 0, the job is finished.

## Coalescence

Immediate coalescence is added as a job leaves memory. Immediate coalescence is achieved my clearing memory and setting all holes to zeros. This default value for memory automatically combines creating bigger holes if there is any small holes (zero) directly around the job that just got removed from memory.

## Compaction

Compaction was added to combine all the holes into one giant hole in memory. There are three different situations where compaction can be called depending on the options selected for the simulation.

1. Every 250 VTUs
2. Every 500 VTUs
3. Memory Request is denied (Job added to the pending list)

## Overview

The simulations have 9 different options that are being compared.

1. First Fit, Compaction Every 250 VTU
2. First Fit, Compaction Every 500 VTU
3. First Fit, Compaction Every Memory Request Denial
4. First Fit, Compaction Every 250 VTU
5. First Fit, Compaction Every 500 VTU
6. First Fit, Compaction Every Memory Request Denial
7. First Fit, Compaction Every 250 VTU
8. First Fit, Compaction Every 500 VTU
9. First Fit, Compaction Every Memory Request Denial

# Memory Placement Strategies

## First Fit

First fit is a memory placement strategy that looks for the first hole that is big enough to place the job in and the places the job there.

## Best Fit

Best fit is a memory placement strategy that looks for the best possible current hole, meaning the hole that would create the smallest hole and places the job there.

## Worst Fit

Worst fit is very similar to best fit except that it finds the spot that would create the worst possible hole, or largest left over hole and places the job there.

# Implementation First Fit vs Best Fit vs Worst Fit

The implementation of First Fit was very easy as you are just required to look through memory until there is place large enough for the job and place the job there. Very simple to implement and requires less overhead than the other to implement as there is no variables needed to store the best location in memory. Overall, this was by far the easy memory placement strategy to implement.

Next, Best Fit and Worst Fit these two memory placement strategies are very similar in one is looking for the best hole and one is looking for the complete opposite. These were almost identical to implement, just a change of a sign. They were not significantly harder to implement than first fit but they were more difficult, in that you had to search the entire memory looking at every memory block to find the best/worst place to put the job.

# Implementation Compaction Scenarios

Compaction is very easily preformed when the performance of the algorithm is very important. The algorithm in the simulation is an O(n^2) algorithm that looks for the first zero and then searches for the next non zero and swaps them and continues to do this until all the zero memory has been combined at the bottom of them memory array.

The three scenario were very easy to implement, the 250 and 500 VTUs just compare the current VTU time and see if it mods with 250 or 500 respectively and then runs the compaction method. The scenario that runs when a memory request is denied was similar in difficulty, each time a job was added to the pending list, the compaction method was called too.

# Performance

In comparing performance many things have to be considered such as jobs completed, pending list size, round robin scheduler size, average hole, fragmentation, storage utilization and average turn-around time, as well as the comparison between the different compaction scenarios for each.

## First Fit – Comparing Compaction Scenarios

### Completed Jobs –Chart A

Comparing the slopes and final values of the completed jobs gives the perception that compaction at memory denial is the better of the scenarios for completed the most jobs, but only by a very small margin. The difference here is very small and just above the standard deviation of the set making it still a valid difference but only barely.

### Pending List Size – Chart B

This graph is showing the number of jobs in the pending list, and almost had no difference at when the pending list is full and the jobs stop entering the system. There are no major improvements between the three scenarios here.

### Round Robin Scheduler Size – Chart C

This graph shows the number of jobs in the ready queue at one time, the higher the number here the more multiprogramming that the system could have. This graph shows that the compaction at 250 VTUs gives you a higher number of jobs on average over the simulation window. However, this there is not many more jobs it is only 1-3 more, which is not a significate difference in the three scenarios.

### Average Hole Size – Chart D

The average hole size starts to show a difference between the three scenarios, the compaction at memory denial has more times when it completely fills up memory and there are no holes. However, it also has the highest points at when memory is the most empty. The other two scenarios are very consistent throughout memory at the level of blank holes in memory around 30-50 KB at any one time.

### Fragmentation – Chart E

Fragmentation has a very similar trend as hole size, in that compaction at memory denial has a much better fragmentation average than any of the other scenarios. Compaction at memory denial has almost no points are when there is a significant amount of fragmentation.

### Storage Utilization – Chart F

Storage Utilization has a much better picture at how efficiently memory is being assigned in compaction at memory denial, there are many points when the memory is perfectly assigned, where the utilization is 100%. There are almost no times when the utilization of the other compaction scenarios are 100%.

### Average Turn-Around Time – Chart G

The average turn-around time of the three scenarios is very similar. They all have a slightly increasing turn-around time.

## Best Fit – Comparing Compaction Scenarios

### Completed Jobs –Chart H

### Pending List Size – Chart I

### Round Robin Scheduler Size – Chart J

### Average Hole Size – Chart K

### Fragmentation – Chart L

### Storage Utilization – Chart M

### Average Hole Size – Chart N

## Worst Fit – Comparing Compaction Scenarios

### Completed Jobs –Chart O

### Pending List Size – Chart P

### Round Robin Scheduler Size – Chart Q

### Average Hole Size – Chart R

### Fragmentation – Chart S

### Storage Utilization – Chart T

### Average Hole Size – Chart U

## Comparing Placement Strategies

### Completed Jobs –Chart V

### Pending List Size – Chart W

### Round Robin Scheduler Size – Chart X

# Selection First Fit vs Best Fit vs Worst Fit

# Improvements Best Best Fit and Worst Worst Fit

After much thought about how to improve the memory placement in the simulation, I decided to look at what would happen if the memory manager would only place a job in the best ever going to be hole and the same for worst. This is what I coined to be best best fit and worst worst fit. This takes out the element in which after a while best and worst fit turned into a modified first fit which I felt would hinder the results of what a true best/worst fit would do. I made these same metrics comparing the previous first/best/worst fit to best best fit and worst worst fit.

## Completed Jobs – Chart F

## Rejected Jobs – Chart G

## Average Hole Size – Chart H

## Fragmentation – Chart I

## Storage Utilization – Chart J

## Improvement Overview

Looking at the benefits of best best fit, while it does provide significant improvements to the rejected job count but other than that it does not really have a benefit that is desirable over best fit or first fit. Best best fit, adds some pretty undesirable metrics that were not measured in this simulation but should be considered before implementation. Best best fit adds a large amount of wait time but not the wait time in the system, the time that a job is waiting to be inserted into memory. In every loop through the simulation the memory manager would wait until almost everything was completed in memory and then insert in the optimal spot in memory. This causes the simulation to not even see many of the jobs that first and best fit would have the chance to see but ultimately reject. Best best fit would not work very well if there was any level of multiprogramming in the system as the CPU would pretty much only have one job to work on.

Worst worst fit, is just the worst of everything but was an interesting exercise to see how close the normal worst fit is at always picking the worst spot and creating the most fragmentation, and it is pretty close on their levels of fragmentation.

## Selection First Fit vs Best Fit vs Worst Fit vs Best Best Fit vs Worst Worst Fit

