## Project – Phase 4

Deliverables will include an updated **software design document**, revision 4, and the **code** (working) for handling the scheduling of processes (jobs) along with Dynamic Partitioning Memory Management enabled, test cases with explanations and screenshots demonstrating your solution for each algorithm and allocation of memory. All capabilities from earlier Phases are assumed to be working and should be used, where possible. Note that you may have to update your Process definition to support for the request and assignment of memory.

# Overview

In this phase – at a minimum your solution needs to able to allocate memory to the processes using a Dynamic Partitioning approach for simulating the Memory Management aspects of an Operating System. **You will be asked as a Team to present and demonstrate your solution for Phase 4**.

# Problem – Phase 4

The development effort of phase 4 focuses on assigning memory to processes in your simulator. Dynamic Partitioning, which can assign a contiguous block of memory to a process based on its requested memory, is to be used. Each process will have its Memory Required specified in the same fashion as its Service time, via the input file or by configuration. Each process must be assigned memory before it can be placed in the Ready State. This will allow it to be quickly assigned to the CPU by your scheduler. A process will hold onto its memory until it completes. Please note that your solution therefore must also handle the situation that processes may arrive, but because of insufficient free memory – cannot be added to the Ready Queue. *But your solution should maximize the number of possible processes in the ready Queue.*  Lastly, your solution must also provide for Compaction, which happens if total free memory is available to add another process to the Ready Queue but there is no single block of free adjacent memory (hole) large enough for the addition of the process. Compaction basically will move all the processes in memory, consolidating all the individual holes into one hole (single block of free memory) at which time the process can then be allocated memory.

In terms of output, your solution must print out the order and when each process completes under the scheduling algorithms (same as previous phase) and must also list the block of memory allocated to each process (starting and ending location) as well as when it was allocated. The same scheduling algorithms and data presented in the previous phases should be presented along with memory assignment aspects for each process.

**Test Data**

It is **required that you construct a smart set of test data that demonstrates compaction and document it in your Test Plan**, in addition it is required that your solution demonstrate the scheduling algorithms with the following data set (Note: it includes a **definition of overall Physical memory and memory required per process**):

Context Switch Penalty = 0

I/O Duration = 5

**Total Physical Memory=4096 MB**

Process 1, arrival time=2, Service Time=6, I/Ofreq=3, **Memory Required=2000 MB**

Process 2, arrival time=3, Service Time=7, I/Ofreq=0, **Memory Required=1000 MB**

Process 3, arrival time=3, Service Time=4, I/Ofreq=5, **Memory Required=3200 MB**

Process 4, arrival time = 4, Service Time=8, I/Ofreq=3, **Memory Required=1000 MB**

Process 5, arrival time=8, Service Time=2, I/Ofreq=1, **Memory Required=100 MB**

Process 6, arrival time=6, Service Time=5, I/Ofreq=0, **Memory Required=400 MB**

**Coversheet (which is required along with all of your material):**

Name(s) of each member

1. Code
2. Test cases with screenshots, include an explanation of what is being verified and why it proves it works.
3. Updated Software Design Document
4. Project Status

**EXTRA CREDIT**

1. Add a graphical interface to show the allocation of memory to processes (as well as holes). We will need to be able to see memory at specific times (when processes arrive or complete, as well as when Compaction happens).
2. Allows processes to surrender memory during I/O events if there is a process waiting that could benefit. This essentially provide for swapping a process out of memory, if and only if another process can be made ready with the swapped process’ memory.