# Main Controller – System Output

**Author: Kenny Parker**

*Note: This document covers the output functionality of the main method*

## Functions

a. printOutput()

Inputs: ( pcb\* runningJob, int totalCycleTracker )

Outputs: void

Function Purpose: Prints the complete output of current simulation state

b. makeString() / currentStatus()

Inputs: ( vector<pcb\*> list )

Outputs: string

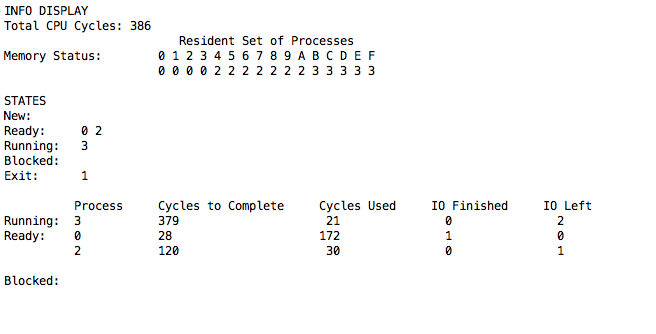
Function Purpose: returns string listing the relevant processes’ PIDs / current data

## Data Structures Used

a. <pcb\*> vectors

## High-Level Description

The main loop tracks when a change is made in the system, when that occurs the printOutput() function prints to the terminal. It is necessary for the visual presentation of the simulation. The printOutput()function is a series of “cout <<” statements related to the headers for three system sections. It also prints the total number of CPU cycles so far at the top of the dialog. Below is a sample output of this function.



## Low-Level Description

The module’s goal is to show the current relevant data. It does so by two parameters passed from the main and by running methods on the system’s various objects to get relevant processes. The main passes the the totalCycleTracker integer which is printed at the top of the output.

Then the resident set of processes are printed by outputting a string of hexadecimal memory addresses and then calling a method within the memory manager that prints a string of the processes in memory.



The printOutput() method then prints lists of the processes in the New, Ready, Running, Blocked, and Exit states. The currently running job’s PID is accessed from the pcb\* pointer called runningJob passed by the main. The other lists of processes are formed by calling the relevant data structure’s method that returns a <pcb\*> vector. Below is an example of such a call.

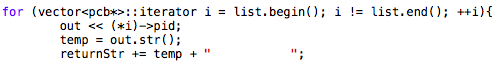


Here is a list of the process states and how to receive a vector of the Process Control Blocks in that state:

* New — hls.getNewProcs()
* Ready — dispatcher.getReadyProcs()
* Running — singular pcb\* parameter, runningJob
* Blocked — interruptManager.getBlockedProcs()
* Exit — dispatcher.getExitProcs()

The returned <pcb\*> vector is then submitted to the function, makeString() to be returned as text.

The makeString() function iterates through the <pcb\*> vector in the following manner: It writes the iterator’s relevant data to a temp string and adds that to the string to be returned for printing.



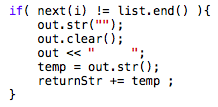
The printOutput() function continues by outputting the current active processes’ relevant data: each process’ Cycles to Complete, Cycles Used, IO Finished, and IO Left. It first prints this data for the singular runningJob if there is a process currently running (determined by if the call to runningJob->pid doesn’t return -1).

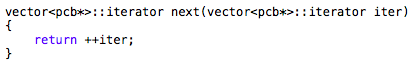
The ready and blocked processes can be numerous and have their relevant data created as a line by a function similar to makeString()called currentStatus(). This function likewise accepts a <pcb\*> vector of the processes in Ready and Blocked states, however it adds to the returned string the numerous data points needed for the currently active processes. Again, an iterator is used to run through the vector.

In detail, the processes’ current state values are computed using the pcb’s following values:

* Cycles to Complete = (\*i)->totalExecutionCycles - (\*i)->currentExecutionCycle
* Cycles Used = (\*i)->currentExecutionCycle
* IO Finished = (\*i)->currentIORequest
* IO Left = (\*i)->totalIORequests - (\*i)->currentIORequest

The currentStatus() function employs a simplistic function, next() that returns a vector<pcb\*>::iterator to check whether the currently writing pcb is the last in the vector. This alters the behavior of the print for the sake of formatting: new space is only added to the next line so long as there is yet another process in that list:







As seen above, the function checks the length of the vector and adds the spacing as seen under “Ready,” but prevents is from being printed after the last line. The entire printOutput() function is designed to provide a clean and logical visual layout for the simulation output.