Output Specification for Project 2:

As you have probably already noticed from my output file, I want you to output a table that reports the following information:

Job #	Running	ReadyToRun	Blocked	Suspended	Total Time	Total Refs	Page Faults
pid 123 *	7	408	10	6	431	23	5
pid 124	100	1426	201	232	1959	170	84
pid 125	120	1900	326	410	2756	155	110
pid 126	50	1118	152	166	1486	49	48
pid 127 *	17	428	29	6	480	110	5
pid 128	150	1779	338	429	2696	461	109
pid 129	50	926	59	56	1091	360	23
pid 130	200	1977	351	533	3061	241	180
pid 131	180	1804	426	523	2933	289	163
pid 132	110	1601	72	96	1879	1119	27
pid 133	190	1969	392	517	3068	214	171
pid 134	75	791	84	121	1071	80	70
pid 135	143	1520	234	390	2287	155	137
pid 136	99	1222	143	94	1558	1108	26
pid 137	199	1813	362	549	2923	213	185

```
Total simulation run time 3099
Total number of jobs 15
Shortest job completion time 431
Longest job completion time 3068
Average job completion time 1978.600
Total Page Faults 1343
Total References 4747
Faults/References percentage 28.29
```

You will note that this looks similar to the output of the previous project except for the following.

- Put a star beside any processes that have Crashed in the Job # column.
- Add a Running column, this output will be identical to the service time in the input file except for processes that have crashed.
- Add a Suspended column that indicates how many clock ticks the process has waited on the suspend queue.
- Add a Total Refs column to indicate the number of memory references the process has made, do NOT count memory references that result in a page fault.
- Add a Page Fault column to indicate the number of page faults a process has generated.
- Report the totals of the last two columns as Total Page Faults and Total References.
- Compute the percentage of memory references that resulted in a page fault (100*Total Page Faults/Total References).

In this assignment, I have given you a set of declarations (page 10 of the specification). You should remove the #define declarations of the following variables and replace them with the following global definitions.

```
int TIME_SLICE = 3;
int MAX_NUMBER_OF_REFERENCES = 25;
int LOCALITY_JUMP = 10;
int NUM_MEM_FAULTS = 5;
```

Your simulator should support 5 command line arguments, one for each of these variables and a verbose output option. You are required to submit an output file with a table using these values of the parameters.

At some point in your careers, you may be asked to write a simulator similar to the one that is assigned in order to evaluate a new architecture that your company is working on. Not only would you write it, but you would evaluate it by changing the parameters of its operation.

In order to help you cope with such a request, you will test your simulator on the data file using the following ranges of values:

2 <= TIME_SLICE <= 4
23 <= MAX_NUMBER_OF_REFERENCES <= 27
9 <= LOCALITY_JUMP <= 11
4 <= NUM_MEM_FAULTS <= 6

Run your simulator once for each possible permutation of these values for a total of 135 simulations. In addition to providing insight into the operation of the system as a function of these parameters, this exercise should really shake the bugs loose. NOTE: You must report the **values of the parameters** and the **minimum Faults/Reference percentage** on the last line of your output file in order to get credit for this part of the assignment. In other words, in addition to the output described on the previous page, you will report 5 values: 4 parameter values and one Faults/ Reference percentage. The parameter values will allow us to verify that your simulator actually generates this value without having to run all of the simulations.

You can use the following perl script to determine the minimum value (assuming your simulator is called sim):

```
#!/usr/local/bin/perl

for ( $i = 2; $i <= 4; $i++ )
   {
   for ( $j = 23; $j <= 27; $j++ )
      {
      for ( $k = 9; $k <= 11; $k++ )
         {
         for ( $l = 4; $l <= 6; $l++ )
            {
            system("sim $i $j $k $l < proj2.dat");
            }
         }
      }
   }
```

PS: Do not forget to comment your code. Robustness, overall code quality and comments count for 20% of the grade !!!!

Point breakdown:
- Correctness: 30%
- Completeness: 20%
- Comments: 20%
- Modularity: 20%
- Command-line args: 10%

You can NOT obtain more than 60% if your program crashes ! You will get 0% for programs that do NOT compile.