```
/*****************************************************************************
/
/      filename:  simulator.cpp
/
/   description:  Implements the simulator to represent Job scheduling,
/                 CPU scheduling, and semaphore processing of an operating syste
m.
/
/        author:  Paladino, Zac
/      login id:  cps346-n1.16
/
/         class:  CPS 346
/    instructor:  Perugini
/    assignment:  PJ #2
/
/      assigned:  February 18, 2009
/           due:  March 11, 2009
/
*****************************************************************************/

#include <iostream>
#include <iomanip>
#include <queue>
#include <list>
#include <vector>
#include <string>
#include <fstream>
using namespace std;

struct Process
{
  string Event;
  string RQ;
  int Time, Job, Memory, RT, RTM, RQT, FTime, STime, IOBurst, IOS, IOB;
  bool started, IOClean;
};

struct Semephore
{
  int value;
    list < Process > SemList;
};

#include "Functions.h"
#include "CPU.h"
#include "Job.h"
#include "IO.h"
#include "Semephore.h"
#include "Output.h"

int
main (int argc, char **argv)
{
  int count = 0, time = 0, memory = 512, CPURQ1 = 100, CPURQ2 =
    300, tot_proc = 0;
  float tot_wait = 0.0, tot_ta = 0.0, ata = 0.0, atw = 0.0;
  bool getcm = false, cpun = true, NOGO = false, ev = false, eve = false, ew =
    false, es = false;
  vector < string > tokens;
  list < Process > JobQ;
  list < Process > RQ1;
  list < Process > RQ2;
  list < Process > CPU;
  vector < Process > Finished;
  vector < Process > IO;
  Semephore Semephores[5];
  string command;
  ofstream out;
  out.open ("out.txt");
```



```
  void *x = NULL;
  x = getline (cin, command);
  tokens = MakeTokens (command, " ");
  while (x != NULL || !JobQ.empty () || !RQ1.empty () || !RQ2.empty ()
         || !CPU.empty () || !IO.empty ()) {
    if (getcm && x != NULL) {
      x = getline (cin, command);
      tokens = MakeTokens (command, " ");
    }

    HandleJob (JobQ, RQ1, tokens, time, ev, eve, getcm, memory, count,
               tot_proc);

    if (!NOGO) {
      HandleIO (IO, CPU, RQ1, tokens, time, getcm, RQ2, Finished, CPURQ1,
                CPURQ2, memory, out, NOGO);


      HandleSem (tokens, Semephores, CPU, RQ1, RQ2, Finished, memory, count,
                 CPURQ1, CPURQ2, getcm, time, out, ew, es);

      EventD (tokens, JobQ, RQ1, RQ2, CPU, Finished, IO, memory, Semephores,
              getcm, time, out, CPURQ1, CPURQ2);
    }

    if (DealCPU (CPU, RQ1, RQ2, Finished, CPURQ1, CPURQ2, memory, time, out)) {
      NOGO = true;
    }
    else {
      NOGO = false;
    }

    if (ev) {
      out << "Event: A  " << "Time: " << time << endl;
      ev = false;
      if (eve) {
        out << "This job exceeds the system's main memory capacity." << endl;
        eve = false;
      }
    }
    if (es) {
      out << "Event: S  " << "Time: " << time << endl;
      es = false;
    }
    if (ew) {
      out << "Event: W  " << "Time: " << time << endl;
      ew = false;
    }

    if (!NOGO) {
      time++;
    }
    count++;

  }
  out << endl;
  out << "The contents of the FINAL FINISHED LIST" << endl;
  out << "---------------------------------------" << endl;
  out << endl;
  out << "Job #  Arr. Time  Mem. Req.  Run Time  Start Time   Com. Time"
    << endl;
  out << "-----  ---------  ---------  --------  ----------  ----------"
    << endl;
  out << endl;
  for (int i = 0; i < (static_cast < int >(Finished.size ())); i++) {
    out << setw (5) << Finished[i].Job << " " << setw (9) << Finished[i].
      Time << " " << setw (9) << Finished[i].
      Memory << " " << setw (8) << Finished[i].
      RT << " " << setw (10) << Finished[i].
```

```
      STime << " " << setw (9) << Finished[i].FTime << endl;
    tot_ta += (Finished[i].FTime − Finished[i].Time);
    tot_wait += (Finished[i].RQT − Finished[i].Time);
  }
  out << endl;
  out << endl;
  out.setf (ios::fixed, ios::floatfield);
  out.precision (3);
  out << "The Average Turnaround Time for the simulation was " << (tot_ta
                                                                  / tot_proc)
    << " units." << endl;
  out << endl;
  out << "The Average Job Scheduling Wait Time for the simulation was " <<
    (tot_wait / tot_proc) << " units." << endl;
  out << endl;
  out << "There are " << memory <<
    " blocks of main memory available in the system." << endl;
  return 0;
}
```