```
/*
 *     File Name                    :   processing.cpp
 *     Primary Author               :   Katie Schaffer
 *     Contributing Author(s)       :   Francesco Polizzi, Jeremy Viner
 *     Date Created                 :   26 April 2016
 *     Date Last Modified           :   6 May 2016
 *
 *     Description          :   Contains processing functions that manage parts of the computer
 *
 */

#include <iostream>
#include <fstream>
#include <iomanip>
#include "simulation_header.h"

#define SUSPEND_TIME 3; // Time required for context-switching

using namespace std;

/* manage_ltq
 * Author: Katelyn Schaffer
 * Other contributors: Francesco Polizzi, Jeremy Viner
 * Date Created: 28 April 2016
 * Last revised: 10 May 2016
 *
 * Description: Manages the longterm queue. Initiates incoming jobs into the queue
 *                  and updates jobs that are currently in the queue
 */
void manage_ltq(longQueue& longterm_queue, job* new_job, FlagContainer& flags) {
        // Handle any current jobs in longterm queue
    if (!longterm_queue.isEmpty()) {
            // Increment wait time for all processes in queue
        longterm_queue.incrementAll();
    }
        // Handle incoming job if the longtern queue is not full
    if (flags.incoming_job && !longterm_queue.isFull()) {
            // Push incoming job to queue
        longterm_queue.add(new_job);
            // Set device enter time
        new_job->lastEnterTime = sys_clock;
            // Remove incoming job flag
        flags.incoming_job = false;
    }

    return;
}

/* manage_stq
 * Author: Katelyn Schaffer
 * Other contributors: Francesco Polizzi, Jeremy Viner
 * Date Created: 28 April 2016
 * Last revised: 10 May 2016
 *
 * Description: Manages the shortterm queue. Updates jobs currently in the queue,
 *                  handles jobs that just finished with the IO device, and
 *                  gets jobs from the longterm queue that need to be moved to the
 *                  shortterm queue
 */
void manage_stq(shortQueue& shortterm_queue, longQueue& longterm_queue,
                IOdevice* io_device, FlagContainer& flags) {
        // Handle any current jobs in shortterm queue
    if (!shortterm_queue.isEmpty()) {
            // Increment wait time for all processes in queue
        shortterm_queue.incrementAll();
    }

        // Handle any job that has just finished with the I/O device
    if (io_device->complete) {
            // Check if the job is finished
        if (io_device->job_finished) {
                // Calculate data
            io_device->process->turnaround = sys_clock - io_device->process->arrival;

                // Collect data
            total_response_time += io_device->process->response;
```

```
            total_productive_time += io_device->process->length;
            total_turnaround_time += io_device->process->turnaround;
            total_switch_time += io_device->process->switching;
            total_stq_wait += io_device->process->time_in_shortQ;
            total_ltq_wait += io_device->process->time_in_longQ;
            total_ioq_wait += io_device->process->time_in_ioQ;

                // Increment counter of total jobs run
            total_jobs_run++;

                // Calculate process's time in system and turnaround time
            io_device->process->time_in_system = sys_clock - io_device->process->arrival;
            io_device->process->turnaround = io_device->process->time_in_system;

                // Flag io device as available
            io_device->available = true;
                // Remove job from the system
            flags.jobs_in_system--;
            io_device->process = NULL;
                // Remove job finished flag
            io_device->job_finished = false;
        }
            // If not finished, place back on shorttterm queue
        else {
                // Check for room in shortterm queue
            if(!shortterm_queue.isFull()) {
                    // Signal no more io interupt
                flags.io_interrupt = false;
                    // Check if IO device has a process and has no just entered
                if (io_device->process != NULL)
                if (io_device->process->lastEnterTime != sys_clock) {
                        // set device enter time
                    io_device->process->lastEnterTime = sys_clock;

                        // Place the process in the shortterm queue
                    shortterm_queue.add(io_device->process);

                        // mark io_device completed
                    io_device->complete = false;

                        // Flag io device as available
                    io_device->available = true;

                        // Reset IO device
                    io_device->process = nullptr;
                }
            }
                // If there is no room in shortterm queue, signal IO interrupt
            else {
                flags.io_interrupt = true;
            }
        }
    } // End handling job finished with IO

        // Check for processes in longterm queue
    if (!shortterm_queue.isNearlyFull() && !longterm_queue.isEmpty()
        && !shortterm_queue.isFull() && longterm_queue.getFront()->lastEnterTime != sys_clock) {
            // Move process from longterm queue to shortterm queue
        shortterm_queue.add(longterm_queue.getNext());
        shortterm_queue.getFront()->lastEnterTime = sys_clock;
    }

    return;
}

/* manage_cpu
 * Author: Katelyn Schaffer
 * Other contributors: Francesco Polizzi, Jeremy Viner
 * Date Created: 28 April 2016
 * Last revised: 10 May 2016
 *
 * Description: Manages the CPU. Handles job processing, handles suspensions, and
 *                  deals with CPU bursts
 */
void manage_cpu(CPU* cpu, shortQueue& shortterm_queue, FlagContainer& flags) {
        // Handle if a process is suspended
```

```
if (cpu->suspended) {
        // Decrement suspend timer
    cpu->suspend_timer--;
        // Increment context switch timer
    cpu->susp_process->switching++;
        // Check if interrupt is complete
    if (cpu->suspend_timer <= 0) {
        flags.interrupt = false;            // Remove interrupt
        cpu->suspended = false;             // Remove suspension
        cpu->process=cpu->susp_process;     // Move suspended process back to CPU
        cpu->susp_process=NULL;             // Process is no longer in suspension
        cpu->processing_stopped=false;      // Signal processing is no longer stopped
    }
        // Otherwise, check if process is in CPU when
        //  interrupt occured
    else if (cpu->susp_process != nullptr) {
            // Update CPU wait counter
        cpu->total_wait++;
            // Flag that processing has stopped
        cpu->processing_stopped = true;
    }
} // End suspend handling

    // If processing has not been halted
if (!cpu->processing_stopped) {

        // Handle interrupt if suspend timer is up
    if (flags.interrupt && cpu->suspend_timer <= 0) {
            // Suspend any process that has the CPU currently
        if (cpu->process != nullptr) {
                // Suspend current process
            cpu->susp_process = cpu->process;
                // Now CPU is free of processes
            cpu->process = nullptr;
        }
            // Reset suspend timer
        cpu->suspend_timer = SUSPEND_TIME;
            // Flag suspension
        cpu->suspended = true;
    }
        // Handle if no interrupt
    else {
            // Handle any process that's in the CPU and check for completion
        if (cpu->process != nullptr) {
                // Update timer of current CPU burst
            cpu->process->cpu_burst[cpu->process->burst_num]--;
                // Track process time in CPU
            cpu->process->time_in_cpu++;

                // Check for completion of burst
            if (cpu->process->cpu_burst[cpu->process->burst_num] <= 0) {
                total_switch_time += 3;
                    // Flag completion
                cpu->complete = true;
                    // Increment burst
                cpu->process->burst_num++;
                    // Reset burst timer
                cpu->timer = 0;
            }
        }
            // Handle any process with completed suspension or get next process
        else {
                // Unsuspend any process that's suspended
            if (cpu->suspended) {
                    // Give suspended process back to the CPU
                cpu->process = cpu->susp_process;
                cpu->susp_process = nullptr;
                    // Increment cpu wait counter
                cpu->total_wait++;
                    // Process is no longer suspended
                cpu->suspended = false;
            }
                // Get next job for the CPU if applicable
            else if (!shortterm_queue.isEmpty() && cpu->ready
                     && shortterm_queue.getFront()->lastEnterTime != sys_clock) {
                    // Give process to the CPU
```

```
                    cpu->process = shortterm_queue.getNext();
                        // set last enter time for job
                    cpu->process->lastEnterTime = sys_clock;
                        // set response time if not already set
                    if (cpu->process->response < 0) {
                        cpu->process->response = sys_clock - cpu->process->arrival;
                    }
                        // Indicate CPU is not ready for more processes
                    cpu->ready = false;
                        // Initialize cpu process timer
                    cpu->timer = 0;
                }
            } // End handling process with completed suspension or next process
        } // End handling no interrupt
    } // End handling if processing has not halted

    return;
}

/* manage_ioq
 * Author: Katelyn Schaffer
 * Other contributors: Francesco Polizzi, Jeremy Viner
 * Date Created: 28 April 2016
 * Last revised: 10 May 2016
 *
 * Description: Manages the IO queue. Updates jobs that are currently in the IO queue,
 *                  and handles jobs that have just finished a CPU burst
 */
void manage_ioq(ioQueue& io_queue, CPU* cpu) {
        // Handle processes in IO queue
    if (!io_queue.isEmpty()) {
            // Increment wait times
        io_queue.incrementAll();
    }

        // Handle any process that the CPU just finished processing
    if (cpu->complete && !io_queue.isFull() && cpu->process->lastEnterTime != sys_clock) {
            // Move process from CPU to IO queue
        io_queue.add(cpu->process);
            // mark enter time for the job we just added
        io_queue.getFront()->lastEnterTime = sys_clock;
            // Reset CPU process num
        cpu->process = nullptr;
            // Indicate CPU is ready for more processes
        cpu->ready = true;
            // Reset cpu_complete flag
        cpu->complete = false;
    }

    return;
}

/* manage_iodevice
 * Author: Katelyn Schaffer
 * Other contributors: Francesco Polizzi, Jeremey Viner
 * Date Created: 28 April 2016
 * Last revised: 10 May 2016
 *
 * Description: Manages the IO device. Simulates serving job IO and retrieves jobs
 *                  from the IO queue
 */
void manage_iodevice(IOdevice* io_device, ioQueue& io_queue, FlagContainer& flags) {
        // Handle if process is in IO device
    if (io_device->process != nullptr) {
            // Update IO timer
        io_device->timer++;

            // Handle if no current interrupt
        if (!flags.io_interrupt) {

                // If finished burst
            if (io_device->timer >= io_device->burst_length) {
                    // Indicate IO complete
                io_device->complete = true;

                total_switch_time += 3;
```

```
                    // Interrupt if more CPU bursts to process
                if (io_device->process->cpu_burst[io_device->process->burst_num] > 0)
                {
                        // Indicate interrupt
                    flags.io_interrupt = true;
                }
                    // Finish up if all bursts are processed
                else {
                    io_device->job_finished = true;
                }
            } // End handling finished burst
        } // End handling process in device
    }
        // If no processes in IO device, handle
        //  any processes in IO queue
    else {
            // Check for processes in IO queue and device availability
        if (!io_queue.isEmpty() && io_device->available
            && io_queue.getFront()->lastEnterTime != sys_clock) {
                // Give IO device to process
            io_device->process = io_queue.getNext();
                // mark the last enter time for io_device job just entered
            io_device->process->lastEnterTime = sys_clock;
                // Update burst length
            io_device->burst_length = io_device->process->io_burst;
                // Reset IO timer
            io_device->timer = 0;
                // Indicate IO device is busy
            io_device->available = false;
        }
    } // End handling IO queue

    return;
}
```