/\*

\* 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;

}