Team: Rick Tilley

Alexander Morehouse

Project: 2 (Bus Terminal)

Class: COP4610

Project Report

In this project our task was to develop a kernel module simulation of an airport shuttle bus. In our simulation, there are 5 terminals lined up, and a shuttle bus traveling between terminals. The bus starts at the middle terminal, and can travel left and right along the axis of airport terminals. The bus has 25 seats and its purpose is to deliver as many passengers as possible to their destined terminals. Each passenger can be a child and occupy 0.5 seat, an adult and occupy 1 seat, or an adult with luggage will occupy 2 seats. The project is developed inside the linux kernel implemented by a custom module that is affected by 3 system calls: start\_shuttle(), issue\_request(char type,int start, int dest), stop\_shuttle(). Any issueing caller can then invoke the start of the shuttle simulation, add waiters at any of the five terminals, and shutdown the bus. At shutdown, the bus delivers the remaining passengers, and then stops without picking up anyone else.

The purpose of this project is to understand how queued process handling and sharing issues can be similar to our bus program. Similar to the cpu sharing to many processes, the bus has to get the people to where they want to go. We don’t want them stepping on each other’s toes, and we want to accelerate the process optimally. In our program there are 5 terminals, and each has a dynamically allocated array that can be used for holding waiting person types. It is a wrap-around array with start and end iterators. This allows it to work as a queue where start++ pops off a waiter, and end++ adds a waiting person. The bus works in a different fashion. It uses a bitmap to map all 25 \* 2 seats. There is an array[50] for seating, as at most 50 children can ride to fill max capacity. Any empty spot in the array[50] will have a corresponding bitmap[i] = -1. The bitmap array holds the person type, or -1 for empty. The other array holds the person struct with information such as start, destination, and type. Any setters or readers are locked at critical sections with a mutex lock. The shuttleloop then determines where to go while picking up and dropping up all in one function call. It then sleeps, and does it again. In order to be optimal, when picking up passengers, if the next person to pick up in the queue doesn’t fit, the algorithm will then look for another waiter in line who has a smaller weight to swap places with. Then a while loop tries to pick up more passengers, until at which point there is no one to pick up or swap places in the queue.

Developing this was most challenging because there is no step through debugger while running in the kernel. Also learning these new conventions for kernel programming was a challenge since we had never attempted that before this project. In all, this program was an interesting learning experience.

**Pseudocode for main shuttle loop:**

shuttleloop()

while (not done)

picknextstate()

sleep(10ms \* timetillchange)

}

thread = 0

bus.action = WAITING

end shuttleloop

picknextstate()

bus.state1 = bus.state2

bus.state2 = {MOVING, timetillchange = 0}

if (anyone to unload)

passenger = 0 < n < 50 : pbitmap[n] = -1

if (passenger.dest == thisterminal)

// clear passenger from bus

// increase stats, accounting variables +-

while (anyone to load that can load)

fatty = terminal.populus[pstart].weight;

if (bus.seats-fatty >= 0)

bus.passengers[openspot] = terminal.populus[pstart++]

// accounting variables +-

else

// try to swap the first person in queue with another smaller person

// adjust timetillchange based on how many loaded/unloaded

if (not parking to load or unload)

// check direction of travel for any waiters or passenger drop offs

// turn around if no reason to go further

// stay put if other direction holds the same outcome

end picknextstate