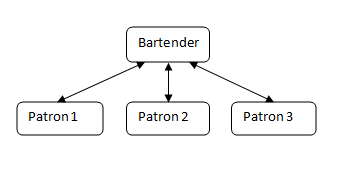
Application Proposal: Bar Program

Overview:

The proposed program will simulate a bar scenario consisting of four separate programs: a bartender and three patrons. The patrons will take turns requesting drinks from the bartender and will yield for a length of time once a drink is received. Once a patron has received the maximum number of drinks that it desires for the evening, the patron will "return home" by terminating.

Entities and Relationships:

 In the bar application, there are four programs, a bartender and three patrons. The following diagram describes their basic relationships:

The first program is unique because there is only one in the application. The bartender program is able to communicate with all of the patrons individually, allowing it to know when a patron is ready for a new drink and to pass a patron their drink. Its method of communication with these processes is through a FIFO structure. The bartender process can take the following actions:

1. Make a drink: when a patron does not have a drink, either at the beginning of the application or after finishing their current drink, they will ask for a new one. The bartender will give the next drink on the menu (an array structure described in the next section) to the patron that has requested one by passing the drink into a FIFO which is used for communication between the bartender and patron processes.

2. Clean the bar: the bar will require periodic cleaning due to normal patron activities. This means that the bartender will be unavailable to fill drink orders for a period of time. During this time, the process will remain idle.

3. Close the bar: once all three patrons have left for the night, the bartender process will close the bar by freeing any memory allocated to the menu structure and terminating the bartender process.

The second program that is used is that of the patron. The only difference between the patrons is that they are created separately and therefore have different process IDs and memory spaces, but otherwise they are created using the same code. They perform identically. The patrons are unique because they are able to communicate with the bartended as previously stated, but not with each other. There is no need for the patrons to share any information since the only actions that they perform are:

1. Order a drink: the patron program alerts the bartender that it has finished with its drink and wishes to have a new one. This means that the patron process will need to yield in order for the bartender process to take over the CPU and make their drink.

2. Consume a drink: once the bartender program finished "making" a drink, the patron process will take the required amount of time to consume the drink. This time is described in the menu structure which is detailed in the next section. In order to consume the drink, the patron will yield for at least the specified amount of time.

3. Return home: when the patron has consumed its maximum number of drinks for the night, the patron will leave the bar. This means that the process will terminate. This will occur at different times depending on the types of drinks that the patron has consumed and the order in which it consumed them.

Additional Structures:

The first additional structure used by the programs in the bar application is the menu structure. The menu is an array of a given size, where the index of the array is the "name" of the drink and the data contained in that index is the time that it takes for the patron is consume the drink. This structure is only accessed in order to know how long the drink will take to consume and is not altered after the start of execution. The bartender process will have access to a variable which will reference the next drink to be given out, since drinks will be given out in a cyclical order. When the bartender has gone through the menu entirely, it will restart at the beginning again. The only process which has access to the menu structure is the bartender; the patrons never interact with it.

The second structure that the program uses is the FIFO which is used for inter-process communication. Only two processes can use the FIFO at a time and one of them is always the bartender. Only one patron can hold a semaphore for the FIFO at any one time. The FIFO is used by the two processes for the transfer of drinks. The bartender places a drink in the FIFO by writing the amount of time that the drink takes to consume into the FIFO, and then the patron can take the drink by reading from the FIFO. Although only the bartender and one other patron can use the FIFO at a time, all four processes share the ability to access to this structure.

Sporadic Processes:

When it is called, the first thing that the bar application does is use OS\_Malloc to allocate space for the menu array structure. The bartender process is started using OS\_Create, before any of the patrons can start. A FIFO will be created for the bartender to communicate with its patrons. Next, the patrons are created one at a time, using OS\_GetParam to retrieve the amount of drinks that each patron wishes to consume in the bar. The patrons will start ordering drinks as soon as they are created by signaling to the bartender that they are ready for a new drink.

From this point, the bartender will not do anything except make drinks when prompted so long as there is at least one patron in the bar. The patrons will order drinks then wait for the appropriate length of time while them consume them , only ordering a new one once the amount of time that the previous one took to consume has passed. Only one patron can order a drink from the bartender at a time, so the patrons must utilize a semaphore when ordering which is locked just before they order and released just after they get their drink.

Once a patron has finished a drink, the patron process will check to make sure that their total drinks consumed has not met or exceeded the amount of drinks that they wanted to consume and, if is has, will trigger a call to OS\_Terminate. This means that the patron will leave the bar and clear any memory or resources such as semaphores that it may have held. If this patron was the last one in the bar, then the bartender will also terminate, but not before closing the bar. This means that the bartender must first free the memory that was used by the menu array before terminating itself in the same way as the patrons.

Periodic Processes:

While there is still at least one patron in the bar, the bartender process will be required to perform a periodic action of cleaning the bar. This means that the bartender will not fill any drink orders during this period of time and any patrons who finish their drinks during this time will need to wait for the bartender to finish cleaning in order to order a new drink. The bartender process itself will not perform any actions during this time, it will simply wait.

Device Processes:

If a button on the board is pressed the bar will close. This means that the patrons will ignore their drink counters and terminate immediately. The termination of all three patrons will automatically trigger the termination of the bartender and thus the closing of the bar. All resources will be released and the application will finish.