Created lab4.h

Added sendbt() def and registercb def

process.h

added two entries (umsg32 sndmsg and bool8 sndflag). Defined PR_SEND
added 3 more entries(int sendhead, int sendtail, and pid32 sendqueue[SENDQSIZE]), defined SENDQSIZE
added another entry (int (* cbfunc)(void)) to store value of callback function

kernel.h

changed QUANTUM to 10

Initialize.c

Initialize sendhead and sendtail to -1 in process initialization. Initialize cbfunc to NULL

receive.c

added code to handle process in send queue

recvclr.c

added code to handle process in send queue

recvtime.c

added code to handle process in send queue

send.c

added sendbt()

mysendqueue.c

created insert_sendq(pid32 receiver, pid32 sender) and dequeue_sendq(pid32 receiver), which insert and dequeue for the sendq of pid32 receiver

resched.c

added code to run callback function

registercb.c

created registercb, stores callback function in processes cbfunc entry.

Part1

A process which is attempting to send using the blocking version, sendbt(), first ensures that maxwait is greater than 0. It then checks to see if the receiving process currently has a message waiting, if it does the process state is set to PR_SEND, the message is stored in the sending process' sndmsg field and sndflag is set to 1. If maxwait time is greater than zero, the process is inserted into the sleepq, and then inserted into the send queue.

The send queue is an additional entry for each process. It is a FIFO circular array queue, inserting into increments the tail by 1 and sets the tail to the pid of the sending process, and dequeue returns to the value of the head, while also decrementing the index of head. If the queue is empty, head and tail are both set to -1.

After being inserted into the send queue, resched is called and the process is blocked until it is ready by a receiving process, or the maxwait time has been reached. When the process resumes, it checks to see if the receiving process has a message, if it does not, then the sending process sets its sndflag to 0, and acts like the send() function. If the receiving process still has a message, something only process who have reached their maxwait time will encounter, then a message is printed stating the process timedout, and TIMEOUT is returned. Otherwise the interrupt mask is restored and OK is returned.

The receive, recvclr, and recvtime functions have all been modified to loop through the receiving process' send queue until it is empty, or it finds a process with a valid message to send. In the loop the sending queue of the current process is dequeued, and the sndflag of that process is checked to see if it is valid. This is how processes that have already timedout but not yet removed from the send queue are removed, since their sndflag is set to 0 when they timeout. Whena process is found with its sndflag set to 1 or the send queue is empty, it exits the loop. Unsleep is called on the sending process id, and if it returns OK the sending process is readied.

Part2

I did not use TS from lab3, I used the default static priority scheduler, and tested my code by ensuring all processes ran at the same priority.

A new entry called cbfunc was created for processes, and in registercb the value of the current process' cbfunc was set to the inputed function func.

The callback function is called after the process is context switched out in resched(). Before calling the callback function, the current process is checked to make sure it has a message waiting to be received, and that the cbfunc is not NULL, if both are true then the callback function is called. This ensures the callback function is executed in the receivers process seamlessly before resuming where it last left off.