OS-Assignment1

Anubhav Palway 2016CS10368

February 2019

1 Part 1

Installations done as instructed in the assignment.

2 Part 2

- To implement syscalls, I defined a index for each syscall in syscall.h
- Defined the function definition in sysproc.c and supporting function in proc.c
- For syscall *toggle*, I made a global variable toggle which switches b/w 0 and 1 when toggle is called.
- For *print_count*, made a char* array which stores the name of the syscall, and *syscall_count* an int array initialized to zero every time toggle is made zero. Call count is stored in this array and it is updated in syscall.c
- *Add* is implemented in sysproc.c and arguments are taken using argint as default argument type in syscall is void. Changed atoi to include negative numbers also.
- *ps* in also written in sysproc.c which calls another function in proc.c which prints all the processes from the process table which are not unused.
- User interface for these functions are defined in user.h and these are linked to syscalls in usys.S

3 Part 3

3.1 Unicast

- *send_msg* and *recv* are implemented by changing the same files as above.
- Message queues are implemented for the communication. Each process has its own message queues.
- Receiver dequeues the message from it's message queue. If it is empty then it changes the flag to 1 telling that it tried to dequeue a message but no message was available and then it goes to sleep. Sender wakes it up after sending the message and then it dequeues again. This time, it will receive the message.
- Sender enqueues the message to receiver's message queues. After this it check if receiver has tried to dequeue the message. If the flag is true then it wakes up the receiver process by calling wakeup function.

3.2 Multicast and Interrupt handler

- In multicast, sender sends the message to all the receivers message queue and tell it to receive the message by sending the interrupt signal.
- *sendsignal* syscall is implemented to send the signal to the receiver process. It pushes the interrupt value (which is always 1 as there is only one interrupt handled in this case) to signal handler stack allocated for each process.

- *sigset* syscall is implemented to give it interrupt handling function which is in the user space. It calls *recv* for each of the receiving process to get the message.
- *checkSignal* function is defined to check if there is any pending interrupt remaining to handle. It runs before the process returns to user space from kernel space from trap function.

3.3 Part 4

- Task of summation of array is divided into 7 child processes. Main process creates 7 child process which calculates the sum of the array allocated to them and it sent the partial sum calculated to the parent process using unicast communication we implemented in the previous part.
- After sending the message the child process goes to sleep. And then the parent process computes the sum of the partial sums sent by each thread. It then computes the mean.
- Mean is then broadcasted to all the child processes using the multicast communication. Parent process wakes up all the child process after sending the sent so that they can compute the variance.
- Each child process the computes sum of $(x - \mu)^2$ for each x belonging to the subarray assigned to this child process.
- After child computes this, it send this back to the parent process and exits.
- Parent then sums all the values received and divide by number of elements to get the variance.