REPORT
Assignment1

SysCalls

Defined the required definition in the right files:
1. syscall.c
2. sysproc.c
3. proc.c
4. proc.h
5. def.h
6. usys.S
7. syscall.h
For toggle I have created a vaiable Trace in proc.c.

For sys_print_count I have added a function in sys_proc.c

```
for(int i=0;i<total_syscalls;i++){
    if(count_sys[i]!=0){
      cprintf("%s %d\n",sys_calls_xv6[i],count_sys[i]);
    }
  }
```

sys_add is simple .

PS require me to write the function in proc.c as ptable can only be accesses from there.

```
void printProcesses()
{
  struct proc *p;
  acquire(&ptable.lock);


  for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
    if(p->state != UNUSED){
      cprintf("pid:%d name:%s\n",p->pid,p->name);
    }
  }

  release(&ptable.lock);
}
```

INTER PROCESS COMMUNICATION

Unicast

I changed the proc struct

```
struct proc {
  uint sz;                     // Size of process memory (bytes)
```

```
  pde_t* pgdir;               // Page table
  char *kstack;               // Bottom of kernel stack for this process
  enum procstate state;       // Process state
  int pid;                    // Process ID
  struct proc *parent;        // Parent process
  struct trapframe *tf;       // Trap frame for current syscall
  struct context *context;    // swtch() here to run process
  void *chan;                 // If non-zero, sleeping on chan
  int killed;                 // If non-zero, have been killed
  struct file *ofile[NOFILE];  // Open files
  struct inode *cwd;          // Current directory
  char name[16];              // Process name (debugging)
----->  struct Queue  message;
};
```

I have declared queue like this.

```
struct Queue{
  uint head;
  uint tail;
  int waiting;
  char MessageQueue[1000][MSGSIZE];//1000 is the Max no. of messages
};
```

Now each process which have its own message Queue.
If a process want to send a message to some pid it will just add to its queue andif a process want to receive a message then it just check its own queue.

Code snippet from sender's code

```
if((p2->message.tail+1)%(1000)==(p2->message.head)%1000){
    return(-1);//Queue of Receiver is full
  }
  for(int i=0;i<MSGSIZE;i++){
    p2->message.MessageQueue[(p2->message.tail)%1000][i]=msg[i];
  }

  // cprintf("Sent from %d\n", send_id);
  p2->message.tail=p2->message.tail+1;
  // cprintf("sendo1 %s\n",p2->message.MessageQueue[(p2->message.head)%1000]);
  // cprintf("rid %d",p2->pid);
  if(p2->message.waiting==1){
    p2->message.waiting=0;
    wakeup1(p2);
  }
```

Code snippet from reiceiver:

```
if((p->message.head)%1000==(p->message.tail)%1000){
    // Empty Queue
```

```
  // Block this process
  p->message.waiting=1;

  sleep(p,&ptable.lock);
  for(int i=0;i<MSGSIZE;i++){
    mess_buf[i]=p->message.MessageQueue[(p->message.head)%1000][i];
  }
  p->message.head+=1;

 }
 else{
  //Making a deep copy

  for(int i=0;i<MSGSIZE;i++){
    mess_buf[i]=p->message.MessageQueue[(p->message.head)%1000][i];
  }
  p->message.head+=1;
  // cprintf("checkthis %s\n",mess_buf);
  // cprintf("mess_bu addrs %d",mess_buf);
 }
```

MultiCast

DISTRIBUTED ALGORITHM

For unicast part
the main process fork the child processes then child process do there corresponind partial sum and send it to parent id.
Then parent receive messages from all the processes and add them.That's it.
Code snippet:

```
for(int i=0;i<p;i++){
        cid=fork();
        if(cid==0){
                //child process
                idx=i;
                int localsum=0;
                int last=(idx + 1) * size/p;
                if(idx==p-1){
                        last=size;
                }
                for (int i =idx*(size/p); i<last; ++i){
                        localsum+=arr[i];
                }
                char *msg = (char *)malloc(8);
                int mypid=getpid();
                itoa(localsum,msg);

                send(mypid,parnet_pid,msg);
                exit();
```

```
        }
    }
```

Main process adds all the result from its children

```
for(int i=0;i<p;i++){
        char *msg_r= (char*)malloc(8);
        recv(msg_r);
        tot_sum+=atoi(msg_r);
}
```

MultiCast part

Thank you