Operating Systems – Assignment 3 Report

Introduction

On this assignment we:

- Implement a basic IPC(Inter-Process Communication) framework allowing processes to send signals to each other, with basic information(an integer).

Storing And Changing The Signal Handler

In order to store the signal handler a new field was added to proc.h:

```
113     sig_handler signalHandler;  // Process Signal Handler
```

And a new system call was added to change the signal handler:

```
547 //---------------------------------------------------
548 //sigset system call implementation
549 //---------------------------------------------------
550 sig_handler sigset(sig_handler new_handler) {
551
552   struct proc *proc = myproc();
553
554   sig_handler old_handler = proc->signalHandler;
555   proc->signalHandler = new_handler;
556   return old_handler;
557 }
558
```

Other changes were made in other files so that the system call would work, just like in the previous assignments.

When fork() is called, the new field for the son is initiated to be equal to the father's, in proc.c:

```
210     np->signalHandler=curproc->signalHandler;
```

When exec() is called, the new process is initiated with a default signal handler equal to -1, in exec.c:

```
22    curproc->signalHandler = (sig_handler)-1; // reset the signal handler
```

Sending A Signal To A Process

In order to store the incoming signals, a new stack field was added to each process, in proc.h:

```
114     struct cstack cstack;       // Stack of signals recieved
115     int ignoreSignals;           // Currently handling signals, or not
```

And a push/pop functions were also implemented in proc.c:

Push:

```
// adds a new frame to the cstack which is initialized with values
// sender_pid, recepient_id and value, then returns 1 on success and 0
// if the stack is full

int push(struct cstack *cstack, int sender_pid, int recepient_pid, int value) {

  struct cstackframe *new_sig;
  for (new_sig = cstack->frames; new_sig < cstack->frames + 10; new_sig++){  // all the frames in cstack
    if (new_sig->used==0){    // if not used we can break
    new_sig->used=1;
        break;
    }
  }
  if (new_sig == cstack->frames + 10) { // no space stuck full
    return 0;
  }
  else {
    new_sig->sender_pid = sender_pid;
    new_sig->recepient_pid = recepient_pid;
    new_sig->value = value;
    new_sig->next = cstack->head;
    cstack->head = new_sig;

  }
  return 1;
}
```

Pop:

```
// removes and returns an element from the head of given cstack
// if the stack is empty, then return 0
struct cstackframe *pop(struct cstack *cstack) {   // pop the head
  struct cstackframe *head;
  head = cstack->head;
  if(head->used == 0)
    return 0;
  head->used = 0;
  cstack->head=head->next;
  return head;
}
```

And only then the required sigsend, sigret system calls were implemented in proc.c:

```
//---------------------------------------------------
//sigsend system call implementation
//---------------------------------------------------
int sigsend(int dest_pid, int value) {

  //current proc
  struct proc *proc = myproc();

  //temp variable for table loop
  struct proc *p;

  //check if pid is valid
  for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
    if (p->pid == dest_pid)
      goto sigsend_dest_pid_found;
  }

  // if dest_pid wan't found, it's not a valid pid. return error
  return -1;

//reciever was found, push the signal to his stack
sigsend_dest_pid_found:

  // if pending signal stack is full, return error
  if (push(&p->cstack, proc->pid, dest_pid, value) == 0)
    return -1;

  //return success
  return 0;
}
```

```
//-------------------------------------------------
//sigret system call implementation
//-------------------------------------------------
void sigret(void) {

  //current proc
  struct proc *proc = myproc();

  //return old trapframe into process
  memmove(proc->tf, &proc->oldTf, sizeof(struct trapframe));

  // process is not handling a signal anymore
  proc->ignoreSignals = 0;
}
```

Also a change was made to the way the kernel handles traps, at trapasm.S:

```
25  trapret:
26    pushl %esp
27    call checkTheSignals  // Check for pending signals, and handle if not already handling.
28    addl $4, %esp
```

It tells the process to check its signals stack, through the function checkTheSignals, found in proc.c:

checkTheSignals part 1:

```
//-------------------------------------------------
//a function for checking which signals need handling
//-------------------------------------------------
void checkTheSignals(struct trapframe *tf){

  //current proc
  struct proc *proc = myproc();

  // if no proc is defined for this CPU, do nothing
  if (proc == 0)
    return;

  // if currently handling a signal, do nothing
  if (proc->ignoreSignals)
    return;

  // if CPU isn't at privilege level 3(in user mode), do nothing
  if ((tf->cs & 3) != DPL_USER)
    return;

  //pop the next frame from stack
  struct cstackframe *poppedCstack = pop(&proc->cstack);

  //if it is empty, no pending signals: do nothing
  if (poppedCstack == (struct cstackframe *)0)
    return;

  // if it is the default signal handler, do nothing
  if(proc->signalHandler == (sig_handler)-1)
    return;
```

checkTheSignals part 2:

```
  //process now handling a signal
  proc->ignoreSignals = 1;

  //back up old trap frame and tell process where to returen when done handling
  memmove(&proc->oldTf, proc->tf, sizeof(struct trapframe));
  proc->tf->esp -= (uint)&invoke_sigret_end - (uint)&invoke_sigret_start;
  memmove((void*)proc->tf->esp, invoke_sigret_start, (uint)&invoke_sigret_end - (uint)&invoke_sigret_start);

  //take parameters given by signal and put into process stack
  *((int*)(proc->tf->esp - 4)) = poppedCstack->value;
  *((int*)(proc->tf->esp - 8)) = poppedCstack->sender_pid;

  //and the address of sigret system call
  *((int*)(proc->tf->esp - 12)) = proc->tf->esp;
  proc->tf->esp -= 12;

  //return into the signal handler
  proc->tf->eip = (uint)proc->signalHandler;

  //free the frame we used
  poppedCstack->used = 0;
}
```

Testing The New Framework

A new user space program called sigsend_test was written to test the framework.

The program:

Part 1:

```c
//-----------------------------------------------------------------------------
//Test for checking sigsend and sigset system calls
//-----------------------------------------------------------------------------

#include "user.h"

#define BUF_SIZE 128

char buf[BUF_SIZE];

//son handler. the handler only prints a message and sends the value he recieved back to sender.
//if 0 is recieved, terminate the son process.
void sonHandler(int fatherPid, int value){
    if (value == 0){
        printf(1, "Son %d recieved a 0 so it exits.\n", getpid());
        exit();
    }
    else{
        printf(1, "Son %d recieved %d and sends it back.\n", getpid(), value);
    }
    sigsend(fatherPid, value);
}

//father handler. the handler only prints what it recieved.
void fatherHandler(int childPid, int value){
    printf(1, "Son %d returned %d as a response.\n", childPid, value);
}

int main(int argc, char *argv[]) {

    //set the handler to be the son's
    sigset((sig_handler)&sonHandler);

    int pid = fork();
    int input;

    //child code
    if(pid==0){
        while(1)
            sleep(5);
        printf(1, "ERROR\n");
        exit();
    }
```

Part 2:

```
44
45      //father code
46      else{
47          printf(1, "Son's pid: %d.\n", pid);
48      }
49
50      //set the handler to be the father's
51      sigset((sig_handler)&fatherHandler);
52
53      //main father loop
54      while (1) {
55
56          sleep(7);
57
58          //get user input
59          printf(1, "Please enter a number (0 for exit): ");
60          gets(buf, BUF_SIZE);
61          if(strlen(buf) == 1 && buf[0] == '\n')
62              continue;
63          input = atoi(buf);
64
65          //clean buffer
66          memset(buf, '\0', BUF_SIZE);
67
68          //send input to son
69          sigsend(pid, input);
70
71          // if user typed 0, finish
72          if (input == 0)
73              break;
74          }
75
76      //wait for son to terminate
77      wait();
78
79      printf(1, "test succesful!\n");
80      exit();
81
82  }
```

And the output when running: