CS360 Final Exam
May 3, 2018
James S. Plank

Instructions

- There are five questions. You must answer all five questions.
- Put your answers on the answer sheets. Do not hand in the exam.
- Put your name and utk email on all of your answer sheets.
- I have put suggested timings for each of these. Use them to gauge your timing.

Things to make your life easier

- I have all sorts of function prototypes on the last page of this exam.
- Do not bother writing down any **#include** statements.
- You do not need to error check any system or library calls with the exception of the **exec** calls.

Question 1 - (5 minutes)

Write the **jassem** assembly code for the C procedure to the right. As always, don't optimize your code.

```
int x(int b)
{
  int *p;

  p = c(b, 1);
  return *p;
}
```

Question 2 - (15 minutes)

A co-worker has written code for an interactive server that interacts with multiple clients, one per thread. The clients can be programs, or they can be using a program like **telnet** or **nc**. There is a part of the code where the server reads two integers from a client connection, uses them to update a shared array, and then writes the array back to the client. The code is to the right.

This code has two serious problems. Neither problem involves the **exit(1)** calls, BTW.

Part A: Identify the two problems -- how they occur and how they manifest as problems.

Part B: Tell me in sufficient detail how you'd fix each problem. "Sufficient detail", means enough to communicate that you know what the solutions are -- if you give me vague wording with buzz-words, then you won't get full credit.

```
typedef struct {
  FILE *fin;
  FILE *fout;
  int data[4096];
  int counter;
  pthread_mutex_t *lock;
} Shared_Info;

void interact(Shared_Info *si)
{
  int x;

  if (fscanf(si->fin, "%d", &x) != 1) exit(1);

  pthread_mutex_lock(si->lock);
  si->data[si->counter%4096] = x;
  si->counter++;

  if (fscanf(si->fin, "%d", &x) != 1) exit(1);
  si->data[si->counter%4096] = x;
  si->counter++;

  fwrite(si->data, sizeof(int), 4096, si->fout);
  fflush(si->fout);

  pthread_mutex_unlock(si->lock);
}
```

CS360 Final Exam - May 3, 2018 - James S. Plank

Question 3 - (25 minutes)

Your job is to implement a data structure called a "twotex." This is like a mutex, only it allows up to two threads to hold the data structure, rather than simply one. It includes these four procedures:

- `void *new_twotex();` -- allocate and initialize the twotex.
- `void lock_twotex(void *t);` -- lock the twotex. Up to two threads can lock the twotex at any one time. If two threads have already locked the twotex, then this procedure should block until the calling thread is one of the two threads that have locked the twotex.
- `void unlock_twotex(void *t);` This unlocks the twotex.
- `void delete_twotex(void *t);` This frees up the memory allocated with `new_twotex`.

Implement these four procedures. You'll also have to define a struct that all four of these use via the `(void *)` To help save you time, you may use the following:

- Abbreviate "pthread_mutex_t" with "PMT".
- Abbreviate "pthread_cond_t" with "PCT".
- Abbreviate "pthread_mutex_lock" with "PML".
- Abbreviate "pthread_mutex_unlock" with "PMU".
- Abbreviate "pthread_cond_wait" with "PCW".
- Abbreviate "pthread_cond_signal" with "PCS".
- Use **new_mutex()** and **new_cond()** as defined below:

```
PMT *new_mutex()
{
  PMT *m;
  m = (PMT *) malloc(sizeof(PMT));
  pthread_mutex_init(m, NULL);
  return m;
}
```

```
PCT *new_cond()
{
  PCT *c;
  c = (PCT *) malloc(sizeof(PCT));
  pthread_cond_init(c, NULL);
  return c;
}
```

CS360 Final Exam - May 3, 2018 - James S. Plank

Question 4 - (50 minutes)

You are on a senior design project sponsored by two local AI companies. One of them has a program, called **AI-Aaron**, which interacts with humans, reading conversation from standard input, and providing converstaion of its own on standard output. The other has a tool, called **AI-Erin**, which does the same thing. **AI-Aaron** runs until its standard input is closed. On the other hand, **AI-Erin** eventually gets sick of conversation, prints out the reason that it is sick of conversation, and exits.

For the senior design project, the two companies want to try out their programs on each other, and figure out the reasons why **AI-Erin** gets sick of conversations.

Of course, the companies don't trust you or each other, so they simply provide you with executable binaries. Your job is to write a program in C that repeatedly (in a **while(1)** loop) creates processes for the two programs to talk to each other, until **AI-Erin** exits. Each time **AI-Erin** exits, your program should print the reason on standard output, and then fire up two new instances at **AI-Aaron** and **AI-Erin**.

Let me give you an example. Your program starts by firing up an instance of **AI-Aaron** and **AI-Erin**, so that they are talking to each other via their own standard inputs and standard outputs. Suppose that the conversation goes:

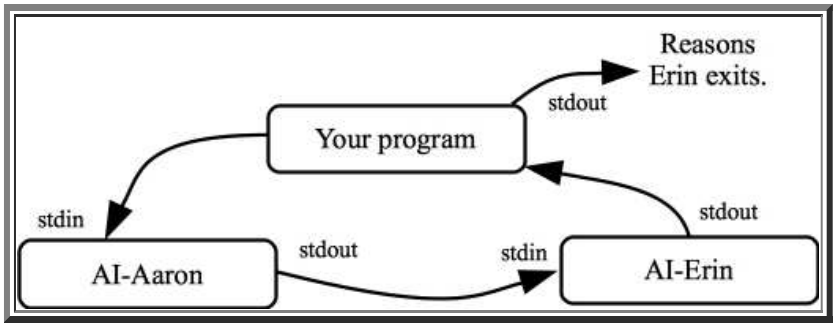
On AI-Erin's standard output, to AI-Aaron's standard input	Hi
On AI-Aaron's standard output, to AI-Erin's standard input	Hi
On AI-Aaron's standard output, to AI-Erin's standard input	I like to program in python!
On AI-Erin's standard output, to AI-Aaron's standard input	You are irrational. I am leaving.
AI-Erin exits.	

Your program should print, "You are irrational. I am Leaving" on its own standard output, and then it should repeat the process by firing up new instances of **AI-Aaron** and **AI-Erin**.

Your program should have the following structure -- when it fires up the **AI-Aaron** and and **AI-Erin** processes, it should have:

- The standard output of **AI-Aaron** going into the standard input of **AI-Erin**.
- The standard output of **AI-Erin** going into your program, so that your program can keep track of the last line. Whenever your program receives a line from **AI-Erin**, it should send it to the standard input of **AI-Aaron**.

Here's a picture:



I have some assumptions and comments for you on the next page.

Assumptions and comments for question 4:

- There are no threads in this program -- you should accomplish everything through system calls.
- You may assume that the lines of text coming from **AI-Aaron** and **AI-Erin** are no more than 99 characters.
- You may not assume that **Ai-Aaron** and **AI-Erin** alternate speaking in their conversations.
- You may assume that **AI-Aaron** and **AI-Erin** exit when their standard inputs are closed.
- Your program should run forever.
- However, your program should not proliferate zombie processes, have memory leaks, or end up exiting with too many open file descriptors.

To make your life easier, you may use the following procedure:

```
void cp(int p[2])
{
  close(p[0]);
  close(p[1]);
}
```

Question 5 - (15 minutes)

On an interview, you are asked by the interviewer: "Explain to me how clients and servers use sockets to communicate with each other over the internet." Your answer should be roughly two paragraphs, and should most definitely include the terms "host", "port", "connection", "client", "server", "file descriptor" and "operating system." You may abbreviate "operating system" with "OS", and "file descriptor" with "FD".

Prototypes of various useful system and library calls

```
int fork();
int wait(int *stat_loc);
int dup2(int fildes, int fildes2);
int pipe(int fildes[2]);

int open(const char *path, int oflag, ...);
int close(int fildes);
ssize_t read(int fd, void *buf, size_t count);
ssize_t write(int fd, const void *buf, size_t count);
off_t lseek(int fd, off_t offset, int whence);

char *strcpy(char *destination, char *source);
char *strdup(char *source);
int   strcmp(char *s1, char *s2);
```

```
typedef void (*sighandler_t)(int);
sighandler_t signal(int signum, sighandler_t handler);

int  setjmp(jmp_buf env);
void longjmp(jmp_buf env, int val);
int  sigsetjmp(sigjmp_buf env, int savesigs);
void siglongjmp(sigjmp_buf env, int val);
```

```
int execl(const char *path, const char *arg, ...);  /* End the argument list with NULL */
int execlp(const char *file, const char *arg, ...);  /* End the argument list with NULL */
int execv(const char *path, char *const argv[]);
int execvp(const char *file, char *const argv[]);
```

Prototypes of Standard IO Library Calls

```
char  *fgets(char *s, int size, FILE *stream); /* Returns NULL on EOF */
int    fputs(const char *s, FILE *stream);     /* Returns EOF when unsuccessful */
int    fflush(FILE *stream);                   /* Returns 0 on success, EOF on failure */
FILE  *fdopen(int fd, char *mode);             /* Returns NULL on failure */

int fgetc(FILE *stream);                       /* Returns EOF on EOF */
int fputc(int c, FILE *stream);                /* Returns EOF when unsuccessful */

size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);
size_t fwrite(void *ptr, size_t size, size_t nmemb, FILE *stream);

int atoi(char *s);   /* Converts a string to an integer - returns zero if unsuccessful */
```

Prototypes from Pthreads

```
typedef void *(*pthread_proc)(void *);
int pthread_create(pthread_t *thread, pthread_attr_t *attr,
                   pthread_proc start_routine, void *arg);

int       pthread_join(pthread_t thread, void **value_ptr);
void      pthread_exit(void *value_ptr);
int       pthread_detach(pthread_t thread);
pthread_t pthread_self();

int pthread_mutex_lock(pthread_mutex_t *mutex);
int pthread_mutex_unlock(pthread_mutex_t *mutex);
int pthread_mutex_init(pthread_mutex_t *mutex, const pthread_mutexattr_t *attr);

int pthread_cond_signal(pthread_cond_t *cond);
int pthread_cond_wait(pthread_cond_t *cond, pthread_mutex_t *mutex);
int pthread_cond_init(pthread_cond_t *cond, const pthread_condattr_t *attr);
```

Prototypes from sockettome.h

```
extern int serve_socket(int port);
extern int accept_connection(int s);
extern int request_connection(char *hn, int port);
```