CS360 Final Exam

Spring, 2023
James S. Plank

The exam was given on Canvas with banks used for some questions. This is an example exam.

Prototypes of various useful system 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);

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

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

Prototypes from libfdr

```
typedef struct inputstruct {
  char *name;               /* File name */
  FILE *f;                  /* File descriptor */
  int line;                 /* Line number */
  char text1[MAXLEN];       /* The line */
  char text2[MAXLEN];       /* Working -- contains fields */
  int NF;                   /* Number of fields */
  char *fields[MAXFIELDS];  /* Pointers to fields */
  int file;                 /* 1 for file, 0 for popen */
} *IS;

IS new_inputstruct(/* FILENAME -- NULL for stdin */);
IS pipe_inputstruct(/* COMMAND -- NULL for stdin */);
int get_line(/* IS */); /* returns NF, or -1 on EOF.  Does not close the file */
void jettison_inputstruct(/* IS */);  /* frees the IS and fcloses the file */

typedef struct dllist {
  struct dllist *flink;
  struct dllist *blink;
  Jval val;
} *Dllist;

Dllist new_dllist();
void free_dllist(Dllist);

void dll_append(Dllist, Jval);
void dll_prepend(Dllist, Jval);
void dll_insert_b(Dllist, Jval);
void dll_insert_a(Dllist, Jval);

void dll_delete_node(Dllist);
int dll_empty(Dllist);

typedef struct jrb_node {
  /* stuff... */
  struct jrb_node *flink;
  struct jrb_node *blink;
  struct jrb_node *parent;
  Jval key;
  Jval val;
} *JRB;

JRB make_jrb();   /* Creates a new rb-tree */

JRB jrb_insert_str(JRB tree, char *key, Jval val);
JRB jrb_insert_int(JRB tree, int ikey, Jval val);
JRB jrb_insert_dbl(JRB tree, double dkey, Jval val);
JRB jrb_insert_gen(JRB tree, Jval key, Jval val, int (*func)(Jval,Jval));

JRB jrb_find_str(JRB root, char *key);
JRB jrb_find_int(JRB root, int ikey);
JRB jrb_find_dbl(JRB root, double dkey);
JRB jrb_find_gen(JRB root, Jval, int (*func)(Jval, Jval));

JRB jrb_find_gte_str(JRB root, char *key, int *found);
JRB jrb_find_gte_int(JRB root, int ikey, int *found);
JRB jrb_find_gte_dbl(JRB root, double dkey, int *found);
JRB jrb_find_gte_gen(JRB root, Jval key, int (*func)(Jval, Jval), int *found);

void jrb_delete_node(JRB node);  /* Deletes and frees a node (but not the key or val) */
void jrb_free_tree(JRB root);  /* Deletes and frees an entire tree */
```

CS360 Final 2023

Question 1 (0 Points)

Please enter your username. For example, my username is **jplank**: ____________-

Question 2 (0 Points)

If you feel the need to make comments on your answers, please do so here, rather than in the answer space for the question. You may leave this blank.

Question 3 (25 points)

For the question, please assume that:

- Pointers are four bytes.
- Pages are 4K bytes (0x1000)
- `malloc()` and `free()` work as described in lecture notes.
- `sbrk(0)` will return 0x7248.
- The free list is doubly linked and NULL terminated.
- The heap starts at address 0x7000.
- The pointer to the head of the free list is at address 0x6200. (It's a global variable, so it is not on the heap).
- Every byte between 0x7000 and the end of the heap is either allocated or free.
- The value at address 0x6200 is 0x7100.
- The value at address 0x7000 is 0x100.
- The value at address 0x7100 is 0x50.
- The value at address 0x7104 is 0.
- The value at address 0x7108 is 0.
- The value at address 0x7200 is 0x248.

Please answer the following questions:

How many bytes are in the heap (allocated and unallocated)? __________
How many total bytes are on the free list? __________
What is the minimum number of allocated chunks in the heap? __________
What is the maximum number of allocated chunks in the heap? __________
Let us suppose I have a `char *` named `p`, whose value is 0x7200, and an integer named `i`. I execute the following loop:

```
for (i = 0; i < 1000000; i++) { *p = 'a'; p++; }
```

What will be the value of `i` when I generate a segmentation violation? You can give me an arithmetic expression here.__________
Suppose that we have the minimum number of allocated chunks in the heap. What is the smallest value of `x` for which `malloc(x)` will require us to call sbrk()? __________
Suppose I call `free(0x7208)`. There are four four-byte quantities that are going to change. For each of these, tell me the address and its new value.
#1 Address__________ #1 Value__________
#2 Address__________ #2 Value__________
#3 Address__________ #3 Value__________
#4 Address__________ #4 Value__________

Question 4 (25 points)

Below is a program where the **main()** routine creates one thread for each word on the command line. Each word is passed as a command to its corresponding thread.

In the questions below, you are given a command line of the program. The output will be a single word on standard output. Your job is to give me all possible outputs of the program. Separate each output with a space. Although the program prints a newline at the end, don't worry about the newline. Just worry about what gets printed.

For example, suppose I want to know the output of:

```
./a.out PP
```

The answer is "AA": A single thread is created that prints 'A' twice.

Here's the program:

```c
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

struct shared {
  pthread_mutex_t lock;
  pthread_cond_t cv1, cv2;
};

typedef struct {
  struct shared *s;
  char *command;
  int id;
} Private;

void *thread(void *arg)                 /* Each thread locks the mutex, and then runs its */
{                                       /* own set of commands.  The commands are characters */
  Private *T;                           /* in the command string.  Each thread has an id, */
  int i;                                /* represented by a character 'A', 'B', 'C', etc. */

  T = (Private *) arg;

  pthread_mutex_lock(&(T->s->lock));
  for (i = 0; T->command[i] != '\0'; i++) {
    if (T->command[i] == 'S') {
      pthread_mutex_unlock(&(T->s->lock));
      sleep(1);
      pthread_mutex_lock(&(T->s->lock));
    }
    if (T->command[i] == 'X') pthread_cond_wait(&(T->s->cv1), &(T->s->lock));
    if (T->command[i] == 'Y') pthread_cond_wait(&(T->s->cv2), &(T->s->lock));
    if (T->command[i] == 'A') pthread_cond_signal(&T->s->cv1);
    if (T->command[i] == 'B') pthread_cond_signal(&T->s->cv2);
    if (T->command[i] == 'P') { printf("%c", T->id); fflush(stdout); }
    if (T->command[i] == 'E') { printf("\n"); exit(1); }
  }
  pthread_mutex_unlock(&(T->s->lock));
  return NULL;
}

int main(int argc, char **argv)
{
  struct shared s;
  int i;
  pthread_t *tids;
  Private *T;
  void *dummy;

  pthread_mutex_init(&s.lock, NULL);   /* Create the lock and two CV's */
  pthread_cond_init(&s.cv1, NULL);
  pthread_cond_init(&s.cv2, NULL);

  T = (Private *) malloc(sizeof(Private)*(argc-1));          /* Create the threads.  Each thread */
  tids = (pthread_t *) malloc(sizeof(pthread_t)*(argc-1));   /* has its own Private struct, with */
  for (i = 0; i < argc-1; i++) {                             /* an id, command and pointer to */
    T[i].s = &s;                                             /* shared data. */
    T[i].id = 'A' + i;
    T[i].command = argv[i+1];
    pthread_create(tids+i, NULL, thread, (void *) (T+i));
  }
  for (i = 0; i < argc-1; i++) pthread_join(tids[i], &dummy);
  printf("\n");
  return 0;
}
```

What are the potential outputs to: `./a.out PP P`? ____________
What are the potential outputs to: `./a.out P P PE`? ____________
What are the potential outputs to: `./a.out XP XP SPAA`? ____________
What are the potential outputs to: `./a.out SPAB XP YP P`? ____________
What are the potential outputs to: `./a.out SPA XPA XPA`? ____________
What are the potential outputs to: `./a.out PXP SABP PYP`? ____________
What are the potential outputs to: `./a.out SPAB XPYP SPAB`? ____________

Question 5 (25 points)

All of these questions have deterministic answers -- there are no "depends". Just an FYI.

How many lines are printed by the following program? _________

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>

int main()
{
  int fd;
  int p;

  fd = open("f1.txt", O_WRONLY | O_CREAT | O_TRUNC, 0666);

  write(fd, "Hi\n", 3);
  p = fork();
  write(fd, "Hi\n", 3);
  close(fd);
  if (p > 0) {
    sleep(1);
    execlp("cat", "cat", "f1.txt", NULL);
  }
  return 0;
}
```

\

How many lines are printed by the following program? _________

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main()
{
  FILE *f;
  int p;

  f = fopen("f1.txt", "w");

  fprintf(f, "Hi\n");
  p = fork();
  fprintf(f, "Hi\n");
  fclose(f);
  if (p > 0) {
    sleep(1);
    execlp("cat", "cat", "f1.txt", NULL);
  }
  return 0;
}
```

CS360 Final 2023

How many lines are printed by the following program? _________

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main()
{
  fork();
  fork();
  fork();
  fork();

  printf("Hi\n");
  return 0;
}
```

The remaining questions are about the following program:

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>

int main(int argc, char **argv)
{
  int p[2];
  int buf[100];
  FILE *f;
  int i;

  for (i = 0; i < 100; i++) buf[i] = i;

  pipe(p);
  if (fork() > 0) {
    if (fork() > 0) {
      sleep(atoi(argv[1]));
      close(p[1]);
      f = fdopen(p[0], "r");
      for (i = 0; i < 100; i++) fread(buf+i, 4, 1, f);
      if (argv[4][0] == 'y') {
        for (i = 0; i < 100; i++) fread(buf+i, 4, 1, f);
      }
      return 0;
    } else {
      sleep(atoi(argv[2]));
    }
  } else {
    sleep(atoi(argv[3]));
  }

  close(p[0]);
  f = fdopen(p[1], "w");
  for (i = 0; i < 100; i++) fwrite(buf+i, 4, 1, f);
  return 0;
}
```

Suppose I call this as follows: `./a.out 6 1 1 y`. What is the total number of times that **read**() is called by all of the processes? _________
Suppose I call this as follows: `./a.out 0 1 4 n`. Can one of these processes generate SIGPIPE? Please answer Y or N. _________
Please enter command line arguments that make one of the processes become a zombie at some point (enter as four words) _________
Suppose I call this as follows: `./a.out 0 1 3 y`. What is the total number of times that **read**() is called by all of the processes? _________
Suppose I call this as follows: `./a.out 10 2 2 n`. What is the total number of times that **write**() is called by all of the processes? _________
Suppose I call this as follows: `./a.out 3 1 0 n`. Can one of these processes generate SIGPIPE? Please answer Y or N. _________
Suppose I call this as follows: `./a.out 8 0 0 n`. Will `buf` contain the numbers from 0 to 99, in sorted order? Please answer Y or N. _________
Please enter command line arguments that make one of the processes become an orphan at some point (enter as four words) _________

Question 6 (25 points)

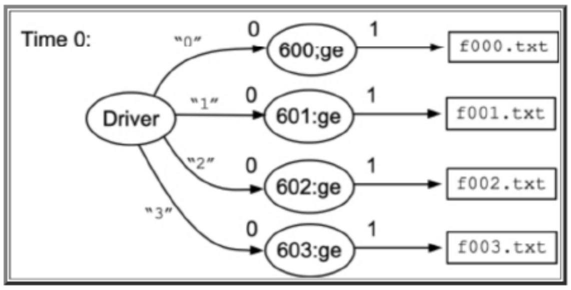
You have software that performs a genetic algorithm on a task that you are trying to optimize. The way it works is that it's an executable program, named **ga**. You will run it multiple times, and each time you give it a different integer seed on standard input. It runs the genetic algorithm, printing output on standard output. Sometimes it runs quickly, but other times it takes a long time, or worse yet, doesn't terminate.

You want to run this 1000 times, using seed values from 0 to 999. You want the output of seed 0 going into the file **f000.txt**, and the output of seed 1 going to **f001.txt**, and so on. Your computer has 4 cores, so you want to make sure that you have 4 processes running at all times.

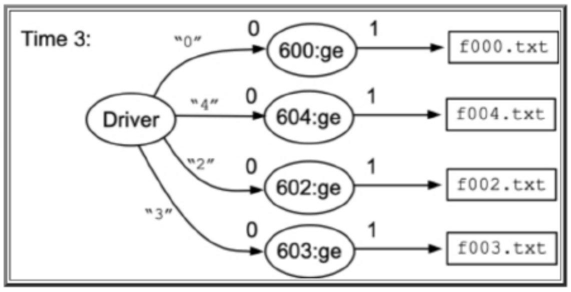
We are going to write a program to manage this. It will **not** be multithreaded. It will call **fork()/execl()/dup()/pipe()/wait()** to run 1000 instances of the **ga** program, with their outputs going to the correct files. It will make sure that there are only 4 processes running at a single time.

Since there are runs that will never terminate, our code should ensure that if an instance of the program is running for an hour or more, that we terminate it. To do this, we'll keep track of when each currently running process has started, and when we discover that it has been running for an hour, we terminate it. By the way, when you want to terminate process **pid**, you use: `kill(pid, SIGINT)`. Our implementation of this at first isn't going to be the best. We'll make it better in a bit.

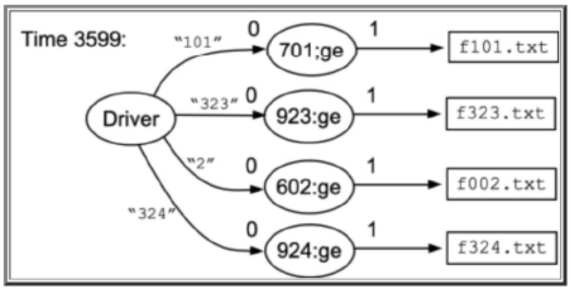
In case this is unclear, here are some pictures. We start at time 0, and our driver has started four **ga** processes. Just for this example, I'm assuming that their process id's are 600 through 603. Here's the picture:



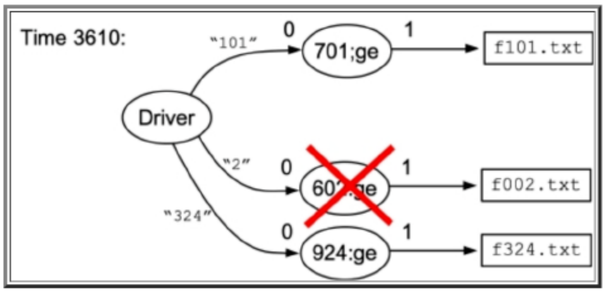
Three seconds later, **ga** process 601 has finished, and the driver has started a new process, which is 604:



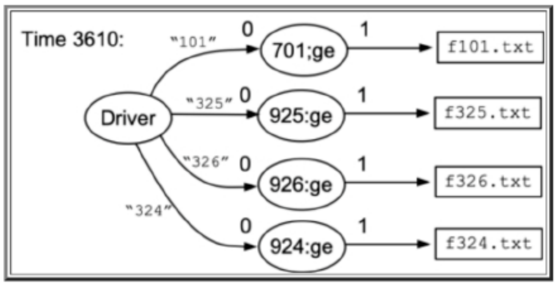
We're 3599 seconds into our program, and **ga** process 602 is still running. Process 701 is taking a while, but as you can see, we've gotten up to seed 324.



When time 3600 comes, we'd like to terminate process 602, but we can't because our program isn't smart enough yet. However, at time 3610, process 923 exits, and at that point, our program detects that process 602 has been running too long, and kills it:



After killing the process, it starts processes 925 and 926:



Part 1 (20 points): Hopefully that gives you the basics of this program. Now, I've written the following **main()** for the program. What I want you to do is write **start_ga()**, which starts off the process with the given seed, and returns the process id of the newly created process, and **kill_slow()**, which terminates any process that has run for at least `TIMEOUT` seconds. You'll note that after you terminate a process, you don't have to do anything fancy with the red-black tree, because when you next call **wait()**, it will return with the newly terminated process.

CS360 Final 2023

Part 2 (5 points): Describe for me what changes you need to make so that you kill processes when *exactly* `TIMEOUT` seconds have elapsed. You're still not allowed to use threads. You don't have to implement this -- just tell me what you need to do. Be specific, please.

```
#include <stdio.h>
#include <time.h>
#include <string.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#include <signal.h>
#include "jrb.h"

#define TIMEOUT 3600
#define SIMUL 4
#define TOTAL 100

void kill_slow(JRB pids)
{
  /* You write this */
}

int start_ga(int seed)
{
  /* You write this */
}

int main()
{
  int pid, dummy, i;
  JRB pids, tmp;

  pids = make_jrb();

  for (i = 0; i < SIMUL+TOTAL; i++) {
    if (i >= SIMUL) {
      kill_slow(pids);
      pid = wait(&dummy);
      tmp = jrb_find_int(pids, pid);
      jrb_delete_node(tmp);
    }
    if (i < TOTAL) {
      pid = start_ga(i);
      jrb_insert_int(pids, pid, new_jval_l(time(0)));
    }
  }

  return 0;
}
```