CS360 Final Exam
May 5, 2016
James S. Plank

Instructions

- There are four questions. You must answer all four questions.
- Put your answers on the answer sheet. Do not hand in the exam.
- Put your name and email on all of your answer sheets.
- Put your major on the first answer sheet.
- I suggest that after you answer question ##, you copy it cleanly on a new answer sheet and turn that new one in.

Things to make your life easier

- Do not bother writing down any **#include** statements.
- You may abbreviate **pthread_mutex_t** as **PMT**.
- You may abbreviate **pthread_cond_t** as **PCT**.
- You may assume that the following two procedures exist -- these will save you some time:

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

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

I have all sorts of function prototypes on the last page of this exam.

CS360 Final Exam - May 5, 2016 - James S. Plank

Question 1 - 20 points

This question is about **malloc()**, as explained in class and in the lecture notes. Specifically, pointers are four bytes, and free nodes contain [size,flink,blink] in that order. The free list is doubly-linked and NULL terminated, with no sentinel.

Suppose memory is below, and the head of the free list is at address `0x10758`.

Address	Value	Address	Value	Address	Value
0x10638	0x38	0x106b0	0x106f4	0x10728	0x10724
0x1063c	0x106d0	0x106b4	0x10670	0x1072c	0x18
0x10640	0x10680	0x106b8	0x18	0x10730	0x10694
0x10644	0x1079c	0x106bc	0x10674	0x10734	0x10760
0x10648	0x1074c	0x106c0	0x0	0x10738	0x106c0
0x1064c	0x10640	0x106c4	0x106f0	0x1073c	0x106c8
0x10650	0x1076c	0x106c8	0x1074c	0x10740	0x18
0x10654	0x10728	0x106cc	0x1063c	0x10744	0x1073c
0x10658	0x1074c	0x106d0	0x50	0x10748	0x1079c
0x1065c	0x106fc	0x106d4	0x10790	0x1074c	0x10658
0x10660	0xc	0x106d8	0x10638	0x10750	0x10694
0x10664	0x10	0x106dc	0x0	0x10754	0x10664
0x10668	0x1075c	0x106e0	0x1069c	0x10758	0x10
0x1066c	0x0	0x106e4	0x1078c	0x1075c	0x10680
0x10670	0x10	0x106e8	0x10710	0x10760	0x0
0x10674	0x106f8	0x106ec	0x0	0x10764	0x10724
0x10678	0x1067c	0x106f0	0x106e8	0x10768	0x28
0x1067c	0xc	0x106f4	0x106bc	0x1076c	0x1073c
0x10680	0x38	0x106f8	0xc	0x10770	0x10710
0x10684	0x10638	0x106fc	0x10690	0x10774	0x1068c
0x10688	0x10758	0x10700	0x106b8	0x10778	0x10794
0x1068c	0x10778	0x10704	0x1c	0x1077c	0x10790
0x10690	0x14	0x10708	0x10738	0x10780	0x10674
0x10694	0x10710	0x1070c	0x10788	0x10784	0x14
0x10698	0x1073c	0x10710	0x0	0x10788	0x106e4
0x1069c	0x106b0	0x10714	0x18	0x1078c	0x10734
0x106a0	0x10680	0x10718	0x10698	0x10790	0x10
0x106a4	0x10728	0x1071c	0x10744	0x10794	0x0
0x106a8	0x10690	0x10720	0x20	0x10798	0x106d0
0x106ac	0x106b0	0x10724	0x1065c	0x1079c	0x106c0

Part 1: On the answer sheet, list for me the chunks of memory on the free list in the order in which they appear on the free list. For each chunk, specify the starting addresss of the chunk, and its size, either in decimal or hexadecimal (use "0x" if you are specifying hex).

Part 2: On the answer sheet, list for me the chunks of memory that have been allocated already using **malloc()**. For each chunk, specify the starting addresss of the chunk and its size. You can specify these in any order.

CS360 Final Exam - May 5, 2016 - James S. Plank

Question 2 - 20 points

On this page and the following page, there are 10 programs. Each of them prints a statement. You need to answer "True" or "False" for each program, according to whether the statement printed by the program is true or false. Assume that the program is compiled to an **a.out** file, and is executed from the shell by simply calling "**./a.out**".

---

q2-p0.c

```
int main()
{
  if (fork() == 0) {
    sleep(1);
    printf("I am an orphan process.\n");
  } else {
    sleep(2);
  }
  exit(0);
}
```

q2-p1.c

```
int main()
{
  int i = 5;

  if (fork() != 0) {
    i += 5;
    sleep(2);
  } else {
    sleep(1);
    printf("In my process, i equals 5.");
  }
}
```

q2-p2.c

```
int main()
{
  int fd;

  fd = open("f1.txt", O_WRONLY | O_CREAT | O_TRUNC, 0666);
  dup2(fd, 1);
  close(fd);

  if (fork() != 0) {
    dup2(2, 1);
    sleep(3);
  } else {
    sleep(1);
    printf("This line will not be printed on the screen.");
  }
}
```

q2-p3.c

```
int main()
{
  int p[2];

  pipe(p);
  if (fork() == 0) {
    dup2(p[0], 0);
    printf("This process will never terminate.\n");
    execlp("cat", "cat", NULL);
    exit(0);
  } else {
    dup2(p[1], 1);
    sleep(1);
    exit(0);
  }
}
```

q2-p4.c

```
int main()
{
  if (fork() == 0) {
    sleep(1);
    printf("I will be a zombie process.\n");
  } else {
    sleep(2);
  }
  exit(0);
}
```

q2-p5.c

```
int main()
{
  char *s = "ABCDEFGHIJKLMNOPQRSTUVWXYZ\n";
  int i;
  int p[2];

  pipe(p);
  close(p[0]);
  if (fork() == 0) {
    for (i = 2; i < 20; i++) dup2(p[1], i);
    printf("I am about to generate SIGPIPE.\n");
    while (1) write(10, s, strlen(s));
  }
}
```

Question 2 - Continued

q2-p6.c

```
int main()
{
  if (fork() == 0) {
    sleep(1);
    printf("I will be a zombie process.\n");
  }
  exit(0);
}
```

q2-p7.c

```
int main()
{
  char *s = "ABCDEFGHIJKLMNOPQRSTUVWXYZ\n";
  if (fork() == 0) {
    printf("I am about to generate SIGPIPE.\n");
    while (1) write(10, s, strlen(s));
  }
  exit(0);
}
```

q2-p8.c

```
int main()
{
  int dummy;

  if (fork() == 0) {
    sleep(1);
    printf("I will be a zombie process.\n");
  } else {
    wait(&dummy);
  }
  exit(0);
}
```

q2-p9.c

```
int main()
{
  if (fork() == 0) {
    sleep(1);
    printf("I am an orphan process.\n");
  }
  exit(0);
}
```

Question 3 - 20 points

Recall from the lecture on the Dining Philosophers that there was a solution where a philosopher would not pick up any chopsticks until both were free. At that point, the philosopher would pick up both chopsticks.

On the next page, there is an implementation of this solution which is similar to what we presented in class. In the solution, you have to define a data structure (**Myphil**) in which you keep the data that your solution needs. You initialize this once, before any threads are created, in **initialize_v()**. Philosopher *i* calls **pickup(i, v)** when he/she wants to eat, and this procedure should not return until philosopher *i* can eat. Philosopher *i* calls **putdown(i, v)** when he/she is done eating.

The implementation doesn't work. Actually, everything about it is ok, except for two things:

- It should not be calling **sleep()** to make the philosopher wait.
- It doesn't have any synchronization or protection of shared variables.

Your job is to change this program so that it implements the solution correctly. You will do this by deleting the **sleep()** call and adding the proper synchronization. That means that you can add variables to **Myphil**, and you can add code to the three procedures.

Note, you can't change any of this code -- your jobs is simply to add code in order to make it work.

CS360 Final Exam - May 5, 2016 - James S. Plank

Question 3 - Continued

Here's how I want you to communicate your answer to me. First, assume that the **sleep(1)** call is gone. That means that line 29 is empty. Next, for each place that you want to add code, specify the line number after which you want to add the code, and then specify the code. For example, if you wanted to print "Hi\n" at the end of **pickup()** and "There\n" at the end of **putdown()**, you would make the answer to the right.

```
After 33:
   printf("Hi\n");

After 43:
   printf("There\n");
```

```
/* Line  1 */  typedef struct {
/* Line  2 */    int num;
/* Line  3 */    int *chopstick_states;  /* 'U' is up, and 'D' is down. */
/* Line  4 */  } Myphil;
/* Line  5 */
/* Line  6 */  void *initialize_v(int phil_count)
/* Line  7 */  {
/* Line  8 */    Myphil *p;
/* Line  9 */    int i;
/* Line 10 */
/* Line 11 */    p = (Myphil *) malloc(sizeof(Myphil));
/* Line 12 */    p->num = phil_count;
/* Line 13 */    p->chopstick_states =
/* Line 14 */       (int *) malloc(sizeof(int)*p->num);
/* Line 15 */    for (i = 0; i < p->num; i++) {
/* Line 16 */      p->chopstick_states[i] = 'D';
/* Line 17 */    }
/* Line 18 */    return (void *) p;
/* Line 19 */  }
/* Line 20 */
/* Line 21 */  void pickup(int i, void *v)
/* Line 22 */  {
/* Line 23 */    Myphil *p;
/* Line 24 */
/* Line 25 */    p = (Myphil *) v;
/* Line 26 */
/* Line 27 */    while (p->chopstick_states[i] == 'U' ||
/* Line 28 */           p->chopstick_states[(i+1)%p->num] == 'U') {
/* Line 29 */      sleep(1);
/* Line 30 */    }
/* Line 31 */
/* Line 32 */    p->chopstick_states[i] = 'U';
/* Line 33 */    p->chopstick_states[(i+1)%p->num] = 'U';
/* Line 34 */  }
/* Line 35 */
/* Line 36 */  void putdown(int i, void *v);
/* Line 37 */  {
/* Line 38 */    Myphil *p;
/* Line 39 */
/* Line 40 */    p = (Myphil *) v;
/* Line 41 */
/* Line 42 */    p->chopstick_states[i] = 'D';
/* Line 43 */    p->chopstick_states[(i+1)%p->num] = 'D';
/* Line 44 */  }
```

CS360 Final Exam - May 5, 2016 - James S. Plank

Question 4 - 20 points

You have brilliant ideas, and you feel the urge to share them with the world. So, you write an IOS app and an Android app called "MyBrilliantIdeas." The idea is to display one of your brilliant ideas on the app, and the idea changes every hour. YOU HAVE SO MANY IDEAS!!!!

Of course, you'll eventually sell advertising and make millions. See how brilliant your ideas are!

Ok, down to business. You need a server. Here's how it's going to work. It will serve a socket on port 30602. Why port 30602? Because the MD5 hash of "MyBrilliantIdeas" is 778a724e0e68a62e03bd96b36eb9e859, and 778a in decimal is 30602. That's a brilliant idea!!

You will have a thread accepting socket connections, and for each connection, you fork off a thread to handle the connection. We'll call that a "connection" thread.

Before I describe connection threads, I'm going to describe the "idea" thread. This thread reads a line from the file **MBI.txt** and stores it into a buffer. It then sleeps for an hour, before it reads another line, overwriting the current line in the buffer. It does this forever. Since your ideas are pithy, you know that each line has a maximum of 499 characters. You don't have to bother closing the open file while the "idea" thread sleeps -- your server is never going to crash.

Now, the connection thread writes the buffer over the socket connection. It then sleeps for an hour, and repeats the process. It doesn't need to check that buffer has actually changed -- if all is working well, the buffer will have changed almost 100% of the time.

Write the server. You need to avoid any race conditions, and you need to handle clients closing the connection gracefully. When you write your code, you don't need **#include** statements. And you can use global variables if you want (only on this question!!)

If you run out of ideas (I'm sure that would never really happen!), simply have the idea thread exit.

You will lose points for memory leaks and for holding a mutex while writing to a socket connection.

Remember the abbreviations and code on the cover page of this exam -- they will make your life easier!

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);
```