CS810 -- Advanced Programming in the UNIX Environment

Final Exam

Instructions: Answer all questions. Answers for the multiple-choice section should be written in the blue book, not circled or otherwise indicated on the exam printout. Write the letter(s) of any answers that you think are correct (not the text of the answer itself); note that there may be more than one correct answer.

Answers for the other sections should also be written neatly in the blue book. You may use more than one blue book if necessary. Write your full name and username on the cover of all used blue books.

This exam is open-book but you may not share another student's book. The only permitted books are the textbooks for the course, ``Advanced Programming in the UNIX Environment'' and ``The C Programming Language''. You may not use any other materials for reference (including your class notes or any electronic devices).

When asked to write code, please include any and all error-checking, error-message generation etc. as you would in a real program. Please read the assignments carefully and think about them before you start writing code. Some of the requirements may not be immediately obvious.

It is recommended that before you write actual code you write down the pseudo-code for the program you are planning to implement. This will give the instructor a chance to give you partial credit when your code does not seem correct.

Multiple choice (2 points per question)

1) How do you determine the creation time of a file?

A. via st_ctime (or st_ctimespec) in struct stat
B. by casting st_ino in struct stat to a time_t
C. implementation dependent / not possible for some systems
D. via creat(2)
E. all of the above
F. none of the above

2) In order to create a new process, you

A. call mkpid(void)
B. call fork(void)
C. call execvp(const char *file, char *const argv[])
D. call system(const char *string)
E. all of the above
F. none of the above

3) A newly created (as explained above) process will have

A. a copy of the stdio buffers previously existing in the parent
B. a unique process ID
C. a copy of the parent's descriptors referencing the same v-node table entries
D. a different parent process ID than the process that created it
E. all of the above
F. none of the above

4) How many times may free be called for a single allocation?

A. Once
B. Twice
C. Once for each time the allocation has been extended with malloc or realloc.
D. The answer is given by the expression (ptr/sizeof(void *)) where ptr is a pointer to the allocation.
E. All of the above.
F. None of the above.

5) How many groups may a single user be in?
A. One
B. Eight
C. Sixteen
D. The answer is implementation-dependent.
E. All of the above.
F. None of the above.

6) Which of the following methods of IPC can be used for two independent processes (ie no common parent/ancestor) on the same host?

A. a socket in the unix domain
B. a socket in the internet domain
C. a fifo
D. a socketpair in the local domain
E. None of the above
F. All of the above

7) What is the effect of 'lseek(fd, 1024, SEEK_END) && write(fd, buf, sizeof(buf))' where fd is a valid file descriptor for the process calling lseek?

A. the file pointer for the descriptor fd is set to the current end of file and the contents of the buffer are appended
B. the file pointer for the descriptor fd is set to the start of the file and contents of the buffer are written over the first 1024 bytes of the file
C. the file pointe for the descriptor fd is moved 1024 bytes beyond the current end of the file and the contents of the buffer are written there, creating a so-called sparse file
D. the answer is implementation dependent
E. all of the above.
F. none of the above.

8) What is the protytpe of the function used by the parent process to retrieve the PID of one of it's children?

A. pid_t getcpid(void)
B. pid_t *getcpids(void)
C. pid_t getppid(void)
D. pid_t *getppid(void)
E. all of the above
F. none of the above

9) 4 points

Elaborate on the effect of running each of these three simple programs. What happens to each process, how many will there be, how do they terminate, ... ?

(a)
```
int main()
{
        while(fork());
        return 0;
}
```

(b)
```
int main()
{
        while(!fork());
        return 0;
}
```

(c)
```
int main()
{
        while(1)
                fork();
        return 0;
}
```

10) 10 points

Consider the files "file1" "file2" "dir/file3" and "dir/file4" with the permissions as indicated below. List all users (jschauma, root, all users in group wheel, all users in group sys, all users in group users, all other users) who may

(a) read each of the files
(b) write to each of the files
(c) remove each of the files

If the answer is non-obvious, explain.

```
$ groups jschauma
users wheel
$ ls -lad .
drwxrwxrwt  4 root  wheel  512 Jul 20 17:35 .
$ ls -la file1 file2 dir
----rw--w-  1 jschauma  wheel  0 Jul 20 17:35 file1
-rw-r--r--  1 root      sys    0 Jul 20 17:33 file2

dir:
total 16
drwxrwx---  2 root      wheel  512 Jul 20 17:34 .
drwxrwxrwt  4 root      wheel  512 Jul 20 17:35 ..
-rw-------  1 jschauma  wheel    0 Jul 20 17:33 file3
-r--rw-r--  1 root      wheel    0 Jul 20 17:33 file4
```

(a) Read-access for "file1":
    Read-access for "file2":
    Read-access for "dir/file3":
    Read-access for "dir/file4":
(b) Write access for "file1":
    Write access for "file2":
    Write access for "dir/file3":
    Write access for "dir/file4":
(c) Able to remove "file1":
    Able to remove "file2":
    Able to remove "dir/file3":
    Able to remove "dir/file4":
    Able to remove "dir":

11) 10 points total:

Consider the following program:

```
 1 #include <sys/types.h>
 2 #include <sys/stat.h>
 3
 4 #include <fcntl.h>
 5 #include <stdio.h>
 6 #include <stdlib.h>
 7 #include <err.h>
 8 #include <errno.h>
 9 #include <unistd.h>
10
11 int
12 main(int argc, char **argv)
13 {
14         pid_t pid;
15         struct flock lock;
16
17         if (write(STDOUT_FILENO, "aaa\n", 4) != 4)
18                 errx(EXIT_FAILURE, "can't write\n");
19
20         lock.l_type = F_WRLCK;
21         lock.l_start = 0;
22         lock.l_whence = SEEK_END;
23         lock.l_len = 4;
24
25         if (fcntl(STDOUT_FILENO, F_SETLK, &lock) < 0)
26                 errx(EXIT_FAILURE, "can't lock\n");
27
28         if ((pid = fork()) == 0) {
29
30                 lock.l_type = F_WRLCK;
31                 lock.l_start = 4;
32                 lock.l_whence = SEEK_END;
33                 lock.l_len = 4;
34
35                 if (fcntl(STDOUT_FILENO, F_SETLK, &lock) < 0)
36                         fprintf(stderr, "child can't lock\n");
37                 else if (write(STDOUT_FILENO, "bbb\n", 4) != 4)
38                         fprintf(stderr, "child can't write\n");
39         } else {
40                 wait();
41                 if (write(STDOUT_FILENO, "ccc\n", 4) != 4)
42                         fprintf(stderr, "parent can't write\n");
43         }
44         return 0;
45 }
```

2.5 points:
(a) What (if any) data is written to stdout?

2.5 points:
(b) What (if any) data is written to stderr?

Now suppose change line 23 as follows:

```
23         lock.l_len = 0;
```

2.5 points:
(c) What (if any) data is written to stdout?

2.5 points:
(d) What (if any) data is written to stderr?

12) 35 points

Write a program 'revline' that, given an arbitrarily large text file, reverses the contents line by line (but does not reverse the words in the file).

Example:

```
$ cat foo
this is line 1
this is line 2
this is line 3
this is line 4
$ revline foo
this is line 4
this is line 3
this is line 2
this is line 1
$
```

13) 25 points

Write a program "check-user-access" that takes two arguments: a pathname and a username. The program will return 0 if the given pathname is readable by the given user; it will return 1 if the given pathname is not readable by the given user (for whatever reason). If the *invoking* user is unable to access the pathname to make this determination, the program writes "Unable to determine user access: %s" to STDERR, where "%s" is replaced with a suitable error message indicating why 'check-user-access' was not able to access the pathname. It then exits with a return code >1.