UNIVERSITY OF TORONTO
Faculty of Arts and Science

AUGUST 2014 EXAMINATIONS

CSC 209 – Software Tools and Systems Programming
Instructor – Bogdan Simion

Duration: 3 hours
No aids allowed

The exam is worth **45%** of your final mark. Note that your mark in the final exam needs to be **at least 40 out of 100** in order to pass the course.

Please answer all questions in the space provided. You may use the blank pages on the reverse side of each page for additional space. In your answers, try to be concise.

The weight of each question is indicated on the side. **Please use the time wisely** - do not spend too much time on one question and end up running out of time for all others. We provided a suggestion on how much time you should spend for each question (including reading it), as a rough guideline, adding up to 2 hours and 30 minutes, leaving 30 minutes for revising. Try to work within these suggested times.

Do not forget to add your name and student number below.

Good luck!

Last name	
First name	
Student number	

Marks

	Q1 T/F	Q2 Progr.	Q3 Bash	Q4 Strings Lists	Q5 Fork Pipes	Q6 Signals	Q7 Sockets	Total
Mark								
Out of	20	24	8	17	14	5	12	100

Question 1 [20 marks] Suggested time: 10 minutes
Indicate for each of the statements, whether they are True or False, in the context of this course. **Circle the correct answer (T/F).**

T / F: The process id of the child process after a fork() call is 0.

T / F: A parent process and its child share the same address space.

T / F: A child process inherits all the file descriptors from its parent.

T / F: If successful, the exec() system call replaces the program being run by a process by a different one.

T / F: A zombie process is a process whose parent is the init process.

T / F: An orphan process is a process terminated by its parent before it has finished its execution.

T / F: Some processes are not descendants of the init process.

T / F: The only way to create a new process (other than init), is to duplicate an existing process.

T / F: The default action for SIGKILL can be changed by installing a signal handler.

T / F: The kill() system call can be used to send any signal to any process or processes.

T / F: A terminating process sends its parent a SIGCHLD signal and will still appear in the process table until the parent accepts its termination status.

T / F: The TCP protocol is more efficient but less reliable than UDP.

T / F: Before sending data over the network, we must convert messages to host byte order.

T / F: The select system call is used for multiplexing I/O events like input from stdin or network connection requests.

T / F: The select system call returns when at least one of the file descriptors in one of the sets is ready for I/O.

T / F: A race condition occurs when multiple processes are trying to do something with shared data and the final outcome depends on the order in which the processes run.

T / F: Operations that modify a shared resource are called critical sections, and must be protected with synchronization mechanisms like semaphores.

T / F: One of the advantages of linked lists over arrays is the dynamic insertion and deletion operations.

T / F: A string's length (as indicated by strlen) and its size (reported by sizeof) are always equal.

T / F: When passing arrays as parameters to functions, the name of an array can be used as a pointer to the first (zero-th) element in the array.

Question 2 [29 marks]

2.1 Bit-wise operations [4 marks] Suggested time: 5 minutes
What output does the following program produce?

```
#include <stdio.h>

int main() {
   int a = 8, b = 7, i;
   b &= 3;
   printf("b=%d\n", b);
   for (i = 0; i < b; i++) {
      a = a >> 1;
      printf("a=%d\n", a);
   }
   return 0;
}
```

Answer:

2.2 Arrays/Pointers relationship [4 marks] Suggested time: 5 minutes
Considering the following piece of code, fill the table below with the values of the array elements after the code is done executing. Be careful with the difference between pointers and values, and pointer arithmetic.

```
int a[4] = {0, 1, 2, 3};
int b = 1;
int *p = a;

p = p + b;
b++;
*p += b;
p = p + b;
*p += 2;
p--;
*p *= 4;
p = p - b;
*p = p - a;
```

a[0]	
a[1]	
a[2]	
a[3]	

2.3 Passing function parameters [4 marks] Suggested time: 5 minutes
What is the output of the following program?

```
#include <stdio.h>

int func(int a, int *b, int *c) {
   a += 5;
   *b += a;
   c = b;
   return a;
}

int main() {
   int x = 5, y = 8, z = 3, t = 0;
   t = func(x, &y, &z);

   printf("x:%d  y:%d  z:%d  t:%d\n", x, y, z, t);
   return 0;
}
```

Answer:

2.4 Memory allocation and memory leaks [8 marks] Suggested time: 10 minutes
There are several errors in the following program. Use the appropriate letter to label the lines of code where the corresponding error occurs.
P - dereferencing a pointer without having allocated memory for it
D - deallocating memory that has already been deallocated
H - deallocating memory that is not located on the heap
M - memory-leak

```
int a=0, b=1, c=2;
int *p, *q, *r;

p = malloc(sizeof(int));
*p = b;
q = &c;
p = &a;
*q = b;
q = malloc(sizeof(10));
*r = a + b + c;
free(p);
r = q;
p = malloc(sizeof(int));
*p = *r;
free(q);
free(r);
free(p);
```

2.5 Structures [4 marks] Suggested time: 5 minutes

What does the following piece of code print?

```
struct Point p1, p2;
struct Point *q1, *q2;

p1.x = 5; p1.y = 10;
p2.x = 1; p2.y = 10;

q1 = &p2;
q2 = &p1;
q1->x += 2;  q1->y += 7;
q2->x += 4;  q2->y += 3;

printf("P1(X,Y) = (%d, %d)\n", p1.x, p1.y);
printf("P2(X,Y) = (%d, %d)\n", p2.x, p2.y);
```

Answer:

Question 3. Bash programming [8 marks] Suggested time: 15 minutes
Write a bash shell script (check_dhcp.sh) that takes 2 arguments and takes actions accordingly.

```
./check_dhcp.sh  <action>  <parameter>
```

The first argument (`action`) indicates the action to be taken, based on its value:

- if its value is **1**, then it finds all the "dhcpd" processes currently running and appends them into a file whose name is specified by "parameter".
- if its value is **2**, then it outputs the last N lines of the file "/etc/dhcpd/dhcpd.conf", where N is the value of "parameter" (you can assume this is a correct numeric value and that the .conf file exists).

Hint: the tail command can be used to retrieve the last N lines of a file:

```
tail -n 10 file.txt
```

will return the last 10 lines of the file file.txt.

Error checking: Your program must make sure that:

- the shell script has exactly 2 commandline arguments, otherwise print a message indicating correct usage ("Usage: check_dhcp.sh <action> <parameter>") and exit.
- the value of the first argument is within the correct range, otherwise print a message indicating the value of this option in unknown.

```
#!/bin/bash
```

Question 4. Strings and Linked Lists [17 marks]

PartA. Strings [6 marks] – Suggested time: 10 minutes
Write a function which determines if a word is a palindrome (returns 1 if true, or 0 if false).
A palindrome is a word that reads the same forward and reversed. For example "Anna" is a palindrom, while "Apple" is not. Assume the word contains only alphabetical characters, either in uppercase or lowercase.
You **must not use any string.h functions (no str* functions, like strlen, strchr, strcpy, etc.)!**
You **may use tolower()** or **toupper()** from <ctype.h>, which convert a char to lower/upper case.

```
#include <ctype.h>
int palindrome(char *word) {

}
```

PartB. LinkedLists [11 marks] - Suggested time: 20 minutes

Assume you have a Linked List structure which contains some information about a student, as follows:

```
typedef struct student {
     int studentNumber;
     char *lastName;
     struct student *next;
}Student;
```

Write a function that traverses a given student list and inserts a new student in alphabetical order (function prototype given below). If the new student has the same last name as another student from the list, insert in ascending order of the student number.

Note: The new student name passed to the function may be deallocated once the function exits, so make sure to create a new copy of the student name when inserting.

```
Student* insertNewStudent(Student* list, int newStudentNo,
                          char* newName) {
```

```
}
```

Question 5. Fork, pipes and exec [14 marks] Suggested time: 30 minutes
Write a C program that implements and executes the following command, without using the appropriate system calls. **You must not use popen() or system()**.

```
head -n 20 < really_big_file.txt | tail -n 10
```

- You can assume that both "head" and "tail" are in the $PATH.
- Skip #include statements; you may use the API sheet provided.
- Make sure you close any file descriptors when not needed anymore.

```
int main() {




}
```

Question 6. Signals [5 marks] Suggested time: 10 minutes

What does the code below produce as output? You may assume that none of the fork() calls fail, and that no external signals are received during the duration of the program.

```
void do_stuff(int c) {
   int s;
   wait(&s);
   printf("Done.\n");
}

void func(int cnum) {
   printf("Woo-hoo: %d\n", cnum);
   exit(cnum);
}

int main() {
   struct sigaction newact;

   newact.sa_handler = do_stuff;
   newact.sa_flags = 0;
   if(sigaction(SIGCHLD, &newact, NULL) == -1) exit(1);

   int i, pid;
   for (i = 0; i < 2; i++) {
       if ((pid = fork()) == -1) {
           perror("Fork");
           exit(1);
       } else if (pid == 0) {
           printf("Child %d created\n", i);
           func(i);
       }
   }

   /* Do some random work */
   int sum=0;
   for(i = 0; i < 10000000; i++)
      sum += 2;

   return 0;
}
```

Answer:

Question 7. Sockets [12 marks] Suggested time: 25 minutes

Implement a "parrot" server, which receives messages from clients and repeats each message back to the client who originally sent it. The server communicates with clients using *reliable* connections, and uses the *I/O multiplexing* model.

Once a client connection is established, the server adds the new communication channel (socket) to a local client array to keep track of it (code for initializing this is provided). Any new messages received from such a client will be repeated back to the client over the corresponding communication channel. The server does *not* timeout while waiting for client connections.

Notes: - Use the API sheet. Syntax doesn't have to be perfect, but **do not write pseudocode!**
- Explain in comments all design decisions (size of queue for pending connections, etc.)
- Some code is already provided to guide you: extracting the port, initializing the array that will keep track of the sockets for clients, initializing file descriptor sets, etc. Have a look over the next page before starting.

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

  // Extract port from commandline
  int portno = atoi(argv[1]);
  // Initialize array of client socket descriptors
  int i, clients[FD_SETSIZE];
  for (i=0; i < FD_SETSIZE; i++)  clients[i] = -1;

  // Fill server info
  struct sockaddr_in serveraddr;
  memset(&serveraddr, 0, sizeof(serveraddr));
  serveraddr.sin_family = AF_INET;
  serveraddr.sin_addr.s_addr = htonl(INADDR_ANY);
  serveraddr.sin_port = htons((unsigned short)portno);

  // Add server operations for incoming client connections
  int serverfd;
   ...
```

```
    // Add serverfd to read set, set maxfd
    fd_set rset, allset;
    FD_ZERO(&allset);
    FD_SET(serverfd, &allset);
    int maxfd = serverfd;

    // Write the main server loop
    for( ; ; ) {









    } //end main server loop

    close(serverfd);
    return 0;
} // end main
```

Page intentionally left blank! Use as extra space for any of the questions (specify which)!

C function prototypes and structs:

```
int accept(int sock, struct sockaddr *addr, int *addrlen)
int bind(int sock, struct sockaddr *addr, int addrlen)
int close(int fd)
int closedir(DIR *dir)
int connect(int sock, struct sockaddr *addr, int addrlen)
int dup2(int oldfd, int newfd)
int execlp(const char *file, char *argv0, ..., (char *)0)
int execvp(const char *file, char *argv[])
int fclose(FILE *stream)
int FD ISSET(int fd, fd set *fds)
void FD SET(int fd, fd set *fds)
void FD CLR(int fd, fd set *fds)
void FD ZERO(fd set *fds)
char *fgets(char *s, int n, FILE *stream)
int fileno(FILE *stream)
pid t fork(void)
FILE *fopen(const char *file, const char *mode)
size t fread(void *ptr, size t size, size t nmemb, FILE *stream);
int fseek(FILE *stream, long offset, int whence); /*SEEK_SET/SEEK_CUR/SEEK_END*/
size t fwrite(const void *ptr, size t size, size t nmemb, FILE *stream);
unsigned long int htonl(unsigned long int hostlong) /* 4 bytes */
unsigned short int htons(unsigned short int hostshort) /* 2 bytes */
char *index(const char *s, int c)
int kill(int pid, int signo)
int listen(int sock, int n)
unsigned long int ntohl(unsigned long int netlong)
unsigned short int ntohs(unsigned short int netshort)
int open(const char *path, int oflag)
/* oflag is O_WRONLY | O_CREAT for write and O_RDONLY for read */
DIR *opendir(const char *name)
int pclose(FILE *stream)
int pipe(int filedes[2])
FILE *popen(char *cmdstr, char *mode)
ssize t read(int d, void *buf, size t nbytes);
struct dirent *readdir(DIR *dir)
int select(int maxfdp1, fd set *readfds, fd set *writefds, fd set *exceptfds,
struct timeval *timeout)
int sigaction(int signum, const struct sigaction *act, struct sigaction *oldact)
/* actions include SIG_DFL and SIG_IGN */
int sigaddset(sigset t *set, int signum)
int sigemptyset(sigset t *set)
int sigprocmask(int how, const sigset t *set, sigset t *oldset)
/*how has the value SIG BLOCK, SIG UNBLOCK, or SIG SETMASK */
unsigned int sleep(unsigned int seconds)
int socket(int family, int type, int protocol) /* family=PF INET, type=SOCK
STREAM, protocol=0 */
int sprintf(char *s, const char *format, ...)
int stat(const char *file name, struct stat *buf)
char *strchr(const char *s, int c)
size t strlen(const char *s)
char *strncat(char *dest, const char *src, size t n)
int strncmp(const char *s1, const char *s2, size t n)
char *strncpy(char *dest, const char *src, size t n)
char *strrchr(const char *s, int c)
int tolower(int c)     /* Converts letter c to lowercase if possible */
int toupper(int c)
int wait(int *status)
int waitpid(int pid, int *stat, int options) /* options = 0 or WNOHANG*/
ssize t write(int d, const void *buf, size t nbytes);
```

Useful structs

```
struct sigaction {
    void (*sa handler)(int);
    sigset t sa mask;
    int sa flags;
}
struct hostent {
   char *h name; // name of host
   char **h aliases; // alias list
   int h addrtype; // host address type
   int h length; // length of address
   char *h addr; // address
}
struct sockaddr in { sa family t sin family;
   unsigned short int sin port;
   struct in addr sin addr;
   unsigned char pad[8]; /*Unused*/
}
```

Shell comparison operators

```
-d filename          Exists as a directory
-f filename          Exists as a regular file.
-r filename          Exists as a readable file
-w filename          Exists as a writable file.
-x filename          Exists as an executable file.
-a, -o               And, or.
-z string            True if empty string
str1 = str2          True if str1 equals str2
str1 != str2         True if str1 not equal to str2
int1 -eq int2        True if int1 equals int2
-ne, -eq, -gt, -lt, -le    For numbers
!=, >, >=, <, <=           For strings
```

Useful shell commands

cat, cut, echo, ls, read, sort, uniq, wc, head, tail
expr match STRING REGEXP
expr ARG1 + ARG2
set (Note: with no arguments set prints the list of environment variables)
ps aux - prints the list of currently running processes
grep (returns 0 if match is found, 1 if no match was found, and 2 if there was an error)
grep -v displays lines that do not match
diff (returns 0 if the files are the same, and 1 if the files differ)
cut -d <delimiter> -f <fieldNumbers>
$0 Script name
$# Number of positional parameters
$* List of all positional parameters
$? Exit value of previously executed command