System and Device Programming
On Off Exam
07.07.2023

Ex 1 (1.5 points)

Suppose that the following program is run using the command
`./pgrm abc 2`
Indicate the possible output or outputs of the program. Note that wrong answers imply a penalty in the final score.

```
#define L 100

int main (int argc, char *argv[]) {
  int i, j;
  char str1[L], str2[L];
  setbuf(stdout,0);
  i = atoi (argv[2]);
  for (j=0; j<i; j++) {
    if (fork () == 0) {
      sprintf (str1, "echo -n [%d]", j);   // “-n” indicates no “new line”
      system (str1);
    } else {
      printf ("{%d}", j);
      sprintf (str1, "%s", argv[1]);
      sprintf (str2, "%d", j);
      execlp (argv[0], "myPgrm", str1, str2, NULL);
    }
  }
  return (0);
}
```

Choose one or more options:

1. ⬜`[1]{1}[0]{0}{1}[1]`
2. ⬜`[0]{0}[0]{0}{1}[1]`
3. ⬜`{0}[0]{1}{0}[1][0]`
4. ⬜`{0}[0]{0}{1}[1][0]`
5. ⬜`[0]{0}[1]{1}{0}[0]`
6. ⬜`{0}[0][1]{1}[0]{0}`
7. ⬜`{0}{0}{1}[1][0][0]`

Ex 2 (1.5 points)

The following pseudo-code segments, executed concurrently by processes P1 and P2 with PID `pid_P1` and `pid_P2`, respectively, indicate which of the following statements are TRUE. Note that wrong answers imply a penalty in the final score.

```
P1
while (1) {
   ...
   kill (pid_P2, SIGUSR1);
   pause ();
   A();
}
```

```
P2
while (1) {
   pause ();
   B();
   ...
   kill (pid_P1, SIGUSR2);
}
```

Choose one or more options:

1. ⬜ It is possible to execute function A() more than one time (without the execution of any B() function in between).
2. ⬜ It is possible to execute function B() more than one time (without the execution of any A() function in between).
3. ⬜ Function A() can be executed before function B().
4. ⬜ They are subject to deadlocks.
5. ⬜ They are subject to starvation.
6. ⬜ Function B() is certainly executed at least one time.
7. ⬜ The first call to function B() is always executed before the first call to function A().
8. ⬜ The first call to function A() is always executed before the first call to function B().

Ex 3 (1.5 points)

Analyze the following code snippet in C++. When the main is executed, indicate how many (standard) constructors, copy constructors, and destructors are called.

```
class C {
   private:
      ...
   public:
     ...
};

int main() {
  C e1;
  C e2 = e1;
  C e3 = *new C;
  return 0;
}
```

Choose one or more options:

1. ⬜ 1 constructor, 2 copy constructors, and 3 destructors.
2. ⬜ 1 constructor, 1 copy constructor, and 2 destructors.
3. ⬜ 2 constructors, 2 copy constructors, and 4 destructors.
4. ⬜ 2 constructors, 2 copy constructors, and 3 destructors.
5. ⬜ 3 constructors, 1 copy constructor, and 3 destructors.
6. ⬜ 3 constructors, 2 copy constructors, and 3 destructors.
7. ⬜ 1 constructor, 2 copy constructors, and 2 destructors.

Ex 4 (1.5 points)

Analyze the following code snippet in C++. Indicate the possible output or outputs obtained by executing the program. Note that wrong answers imply a penalty in the final score.

```
auto lambda = []( std::string h )->bool{
   return ( h != "-" && h != "." );
};

int main() {
   std::string s("123.456.789-00");
   std::vector<std::string> num;
   for (int i = 0; i < s.length() ; i++) {
      num.push_back( s.substr(i, 1) );
   }
   cout << s << "#";
   for( auto z : num ){ if (lambda(z)) std::cout << z; }; std::cout << '\n';
   return 0;
}
```

Choose one or more options:

1. ⬜ The program displays the sequence "`123.456.789-00#`"
2. ⬜ The program displays the sequence "`123.456.789-00#..-`"
3. ⬜ The program does not run as there is a bug.
4. ⬜ The program displays the sequence "`123.456.789-00#12345678900`"
5. ⬜ The program displays the sequence "`123.456.789-00#123.456.789-00`"
6. ⬜ The program displays the sequence "`12345678900#12345678900`"

Ex 5 (1.5 points)

Analyze the following code snippet and suppose the user introduces the following strings from standard inputs.
`The input is: A An And AND The`
Indicate which of the following statements are correct. Note that wrong answers imply a penalty in the final score.

```
#include <map>
#include <set>
...

map<string, size_t> c;
set<string> s = {"The","But","And","Or","An","A",
                 "the","but","and","or","an","a"};
string word;
while (cin >> word)
  if (s.find(word)==s.end())
    ++c[word];
for (const auto &w : c) {
  cout << w.first << "=" = << w.second << " ";
}
```

Choose one or more options:

1. ⬜ The code displays: "`The input is: A An And AND The`"
2. ⬜ The code displays: "`2 1 1 1 1 1 1 2`"
3. ⬜ The code displays: "`AND=1 input=1 is:=1`"
4. ⬜ The code displays: "`input=1 is:=1 AND=1`"
5. ⬜ The code displays: "`input= is=1 AND=1`"
6. ⬜ The code displays: "`AND=1 input=1 is=1`"
7. ⬜ The code displays: "`The=2 input=1 is:=1 A=1 An=1 And=1 AND=1 The=2`"

Ex 6 (1.0 points)

Refer to the use of signals in a UNIX/Linux environment. Indicate which of the following statements are correct. Note that incorrect answers imply a penalty in the final score.

Choose one or more options:

1. ⬜ By using the system-call `signal()`, you can decide to ignore ANY type of signal.
2. ⬜ Inside a signal handler, only reentrant functions should be used.
3. ⬜ The reception of a signal by a process ALWAYS causes the termination of the process.
4. ⬜ The use of the pair of functions `kill()` and `pause()` or of the semaphore primitives `sem_post()` and `sem_wait()` are EQUIVALENT from the point of view of process synchronization.
5. ⬜ The reception of some signal types (for instance, `SIGKILL`) cannot be ignored.
6. ⬜ The execution of a signal handler after the reception of a signal by a process can lead to race conditions.
7. ⬜ A signal can be masked only when sent with the system call `kill()`, not the shell command `kill`.
8. ⬜ Both the shell command `kill` and the system call `kill()` can terminate a process.
9. ⬜ The system-call `kill()` always terminates a process; the shell command `kill` does not.
10. ⬜ The system-call `kill()` sends a signal to a process.

Ex 7 (1.0 point)

Suppose that a process becomes a "zombie”. Indicate which of the following statements are correct. Note that wrong answers imply a penalty in the final score.

Choose one or more options:

1. ⬜ The PCB of the process is still present in the memory.
2. ⬜ The zombie process performed a wait.
3. ⬜ The parent of the zombie process performed a wait.
4. ⬜ The parent of the zombie process has not made a `wait` (or a `waitpid`) yet.
5. ⬜ The process does not have a parent process.
6. ⬜ The PCB of the parent process will be deleted only after the child process performs a wait.
7. ⬜ The process was undoubtedly inherited by the process “init”.
8. ⬜ Its PCB will be deleted only after its parent performs a `wait` or a `waitpid`.

Ex 8 (1.5 points)

Suppose to execute the following program. Indicate which are the possible outputs generated by the program. Note that wrong answers imply a penalty in the final score.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include "pthread.h"
#include "semaphore.h"

sem_t *sa, *sbc, *sd;
int na, nbc, nd;

static void *TA ();
static void *TB ();
static void *TC ();
static void *TD ();

int main (int argc, char **argv) {
  pthread_t th;
  sa = (sem_t *) malloc (sizeof(sem_t));
  sbc = (sem_t *) malloc (sizeof(sem_t));
  sd = (sem_t *) malloc (sizeof(sem_t));
  na = nbc = nd = 0;
  sem_init (sa, 0, 1);
  sem_init (sbc, 0, 0);
  sem_init (sd, 0, 0);

  setbuf(stdout, 0);

  pthread_create (&th, NULL, TA, NULL);
  pthread_create (&th, NULL, TB, NULL);
  pthread_create (&th, NULL, TC, NULL);
  pthread_create (&th, NULL, TD, NULL);

  pthread_exit(0);
}

static void *TA () {
  pthread_detach (pthread_self ());

  while (1) {
    sem_wait (sa);
    printf ("A");
    na++;
    if (na==1) {
      sem_post (sa);
    } else {
      na = 0;
      sem_post (sbc);
    }
  }

  return 0;
}

static void *TB () {
  pthread_detach (pthread_self ());

  while (1) {
    sem_wait (sbc);
    printf ( "B");
    nbc++;
    if (nbc==2) {
      sem_post (sd);
    }
    sem_post (sbc);
  }

  return 0;
}

static void *TC () {
  pthread_detach (pthread_self ());

  while (1) {
    sem_wait (sbc);
    printf ( "C");
    nbc++;
    if (nbc==2) {
      sem_post (sd);
    }
    sem_post (sbc);
  }

  return 0;
}

static void *TD () {
  pthread_detach (pthread_self ());

  while (1) {
    sem_wait (sd);
    if (nd==0) {
      sem_wait (sbc);
    }
    printf ("D");
    nd++;
    if (nd==1) {
      sem_post (sd);
    } else {
      printf ("\n");
      nbc = nd = 0;
      sem_post (sa);
    }
  }

  return 0;
}
```

Choose one or more options:

1. ⬜ AAABCDD
2. ⬜ AABCDD
3. ⬜ AABCDDD
4. ⬜ AACBDD
5. ⬜ AAABDD
6. ⬜ AABCBCDD
7. ⬜ AADD
8. ⬜ AACCBBDD
9. ⬜ ABCD
10. ⬜ ACBD

Ex 9 (1.0 point)

The following code snippet uses a condition variable using the POSIX library, shared between two threads, T1 and T2. Indicate which of the following statements are correct. Note that more than one response can be correct and that incorrect answers may imply a penalty on the final score.

```
pthread_mutex_t m = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t cv = PTHREAD_COND_INITIALIZER;
int done = 0;

void *T1 () {
  pthread_mutex_lock (&m);
  while (done == 0)
      pthread_cond_wait (&cv, &m);
  pthread_mutex_unlock(&m);
  return 0;
}
```

```
void *T2 () {
  pthread_mutex_lock (&m);
  done = 1;
  pthread_cond_broadcast (&cv);
  pthread_mutex_unlock (&m);
  return NULL;
}
```

Choose one or more options:

1. ⬜ If T1 runs before T2, it will release the mutex `m`, and it will sleep on the condition variable cv with the system-call `pthread_cond_wait`.
2. ⬜ If T2 runs before T1, the signal issued with `pthread_cond_broadcast` on the condition variable `cv` will be stored by the operating system such that T1 will not wait.
3. ⬜ If there are more instances of T1 waiting on the condition variable `cv` simultaneously, only one will be woken up by T2.
4. ⬜ If T2 runs twice without T1 running, the first signal issued with `pthread_cond_broadcast` will be lost, and the second enqueued by the operating system.
5. ⬜ The mutex `m` can be substituted by a semaphore.
6. ⬜ The system-call `pthread_cond_wait` can be substituted by `pthread_cond_timedwait`.
7. ⬜ In the previous scheme, T1 can wait forever.
8. ⬜ In the previous scheme, T2 can wait forever.

Ex 10 (1.0 point)

A function is defined as follows:

```
void foo (int v, int *p, int &r);
```

Indicate which of the following statements are correct. Note that more than one response can be correct and that incorrect answers may imply a penalty on the final score.

Choose one or more options:

1. ⬜ The function call `foo(24,36,12)` is correct.
2. ⬜ The function call `foo(24,&i,&j)` is correct.
3. ⬜ The function call `foo(24,&i,j)` is correct.
4. ⬜ The value of `r` cannot be modified.
5. ⬜ The value referenced by `p` cannot be modified.
6. ⬜ With the parameter by value `v`, we must copy a memory value.
7. ⬜ With the parameter by address `p`, we must copy a memory value.
8. ⬜ With the parameter by reference `r`, we must copy a memory value.
9. ⬜ Inside the function `f`, the operation `p=&r` is illegal.
10. ⬜ Inside the function `f`, the operation `*p=v` is illegal.

Ex 11 (1.0 point)

Consider the C++ RAII paradigm. Indicate which of the following statements are correct. Note that more than one response can be correct and that incorrect answers may imply a penalty on the final score.

Choose one or more options:

1. ⬜ A resource can be allocated using the `new` operator.

2. ⬜ A resource can be allocated using the `new (nothrow)` operator.
3. ⬜ With RAII, we need to free the memory explicitly with the operator `delete`.
4. ⬜ With RAII, the life cycle of a resource is bound to the lifetime of an object.
5. ⬜ With RAII, `share_ptr` should never be used.
6. ⬜ With RAII, standard pointers should never be used.
7. ⬜ With RAII, `share_ptr` is associated with a counter to understand when to erase an object.
8. ⬜ With RAII, `weak_ptr` manipulates the counter associated with the pointer as the `share_ptr`.
9. ⬜ In the definition `share_ptr<T> p;` T indicates the type of the object the pointer will be pointing to.
10. ⬜ If `ptr` is a pointer, the code `delete ptr; ptr=nullptr;` can be used to (at least partially) avoid dangling pointers.

Ex 12 (1.0 point)

Referring to thread pools, indicate which of the following statements are correct. Note that incorrect answers may imply a penalty on the final score.

Choose one or more options:

1. ⬜ Manage a specific number of worker threads.
2. ⬜ Can be implemented using a semaphore throttle.
3. ⬜ Use a queue of tasks from which the workers get new jobs to run.
4. ⬜ Are used to avoid to re-create threads every time.
5. ⬜ Use a semaphore to fix the maximum amount of running threads.
6. ⬜ May have a callback function passed to the running thread to do its job.
7. ⬜ May follow a producer and consumer scheme.
8. ⬜ May follow a Reader and Writer scheme.
9. ⬜ May be used to avoid over-subscription.