System and Device Programming
On Off Exam
04.09.2023

Ex 1 (1.5 points)

Suppose that the following program is run using the command
./pgrm 2
Indicate which are the possible outputs generated. Note that more than one response can indeed be correct and that incorrect answers may imply a penalty on the final score.

```
#define N 100

int main (int argc, char *argv[]) {
  int n;
  char str[N];
  n = atoi (argv[1]);
  setbuf(stdout,0);
  while (n>0 && !fork()) {
    fprintf (stdout, "F");
    if (fork()) {
      fprintf (stdout, "E");
      sprintf (str, "%d", n-1);
      execlp (argv[0], argv[0], str, NULL);
    } else {
      sprintf (str, "echo -n S");
      system (str);
    }
    n--;
  }
  return 1;
}
```

Choose one or more options:

1. ⬜ FFFEEESSS
2. ⬜ FEFEFESSS
3. ⬜ FEFESFESS
4. ⬜ FSFSFSEEE
5. ⬜ FESFSEFES
6. ⬜ FESFEEFSS
7. ⬜ FSEFSSFEE
8. ⬜ FSFSEEFSE

Ex 2 (1.5 points)

Suppose that the following program is run using the command
./pgrm 4
Indicate which are the possible outputs generated. Note that more than one response can indeed be correct and that incorrect answers may imply a penalty on the final score.

```
void *t1 (void *a){
  pthread_t th1, th2;
  int n1, n2, *p;
  p = (int *) a;
  if (*p>0) {
    printf ("%d ", *p);
```

```
    n1 = *p - 1;
    pthread_create (&th1, NULL, t1, (void *) &n1);
    n2 = *p - 2;
    pthread_create (&th2, NULL, t1, (void *) &n2);
  }
  pthread_join (th1, NULL);
  pthread_join (th2, NULL);
  pthread_exit (NULL);
}

int main(int argc, char **argv) {
  pthread_t th;
  int n = atoi (argv[1]);
  pthread_create (&th, NULL, t1, (void *) &n);
  pthread_join (th, NULL);
  pthread_exit (NULL);
}
```

Choose one or more options:

1. ⬜ 4 2 2 3 1 1 1
2. ⬜ 4 3 2 1 2 1 1
3. ⬜ 4 2 1 3 1 1 2
4. ⬜ 4 3 2 1 1 2 1
5. ⬜ 4 2 2 1 3 1 1
6. ⬜ 4 2 3 1 1 1 2
7. ⬜ 4 3 1 1 2 2 1
8. ⬜ 4 3 2 1 2 1 1

Ex 3 (1.5 points)

Suppose to run the following program. Indicate which are the possible outputs generated. Note that more than one response can indeed be correct and that incorrect answers may imply a penalty on the final score.

```
typedef struct cond_s {
  pthread_mutex_t lock;
  pthread_cond_t cond;
  int count;
  int flag;
} cond_t;

static void *TA (void *args) {
  cond_t *cond_d = (cond_t *) args;
  while (1) {
    pthread_mutex_lock (&cond_d->lock);
    cond_d->count--;
    printf ("%d ", cond_d->count);
    if (cond_d->count <= 0) {
      cond_d->flag = 1;
      pthread_cond_signal (&cond_d->cond);
      pthread_mutex_unlock (&cond_d->lock);
      break;
    }
    pthread_mutex_unlock (&cond_d->lock);
  }
  pthread_exit(0);
```

```
}

static void *TB (void *args) {
  cond_t *cond_d = (cond_t *) args;
  pthread_mutex_lock (&cond_d->lock);
  while (cond_d->flag == 0) {
    pthread_cond_wait (&cond_d->cond, &cond_d->lock);
    printf ("(%d) ", cond_d->count);
    cond_d->count--;
  }
  pthread_mutex_unlock (&cond_d->lock);
  pthread_exit(0);
}

int main () {
  cond_t cond_d;
  pthread_t tid1, tid2;
  setbuf (stdout, 0);

  pthread_mutex_init (&cond_d.lock, NULL);
  pthread_cond_init (&cond_d.cond, NULL);
  cond_d.count = 10;
  cond_d.flag = 0;

  pthread_create (&tid1, NULL, TA, (void *) &cond_d);
  pthread_create (&tid2, NULL, TB, (void *) &cond_d);
  pthread_join (tid1, NULL);
  pthread_join (tid2, NULL);
  printf ("[%d]", cond_d.count);
  pthread_exit(0);
}
```

Choose one or more options:

1. ⬜ 9 8 7 6 5 4 3 2 1 [-1]
2. ⬜ (9) 8 (7) 6 (5) 4 (3) 2 (1) 0
3. ⬜ 9 (8) 7 (6) 5 (4) 3 (2) 1 (0)
4. ⬜ 9 8 7 6 5 4 3 2 1 0 (0) [-1]
5. ⬜ 9 8 7 6 5 4 3 2 1 0
6. ⬜ 9 8 7 6 5 4 3 2 1 0 [0]
7. ⬜ 9 8 7 6 5 4 3 2 1 [0]

Ex 4 (1.5 points)

Suppose to run the following program. Indicate which are the possible outputs generated. Note that more than one response can indeed be correct and that incorrect answers may imply a penalty on the final score.

```
int main() {
  int i, j;
  vector<int> v{0,1,2,3,4,5,6};
  auto l = [&](int i){ swap(v[i], v[v.size()-1-i]); };

  for (i=0, j=v.size()-1; i<j; i++, j--){
    cout << v[i] << " ";
    l(i);
  }
```

```
  cout << "# ";
  for(auto e: v){
    cout << e << " ";
  }

  return 1;
}
```

Choose one or more options:

1. ⬜ 1 2 3 # 6 5 4 3 2 1 0
2. ⬜ 6 5 4 3 2 1 0 # 6 5 4 3 2 1 0
3. ⬜ 0 1 2 # 6 5 4 3 2 1 0
4. ⬜ 6 5 4 # 6 5 4 3 2 1 0
5. ⬜ 0 1 2 3 4 5 6 # 6 5 4 3 2 1 0
6. ⬜ 0 1 2 # 0 1 2 3 4 5 6
7. ⬜ 6 5 4 # 6 5 4 3 2 1 0

Ex 5 (1.5 points)

Suppose to run the following program. Indicate which are the possible outputs generated. Note that more than one response can indeed be correct and that incorrect answers may imply a penalty on the final score.

```
int main() {
  vector<int> v;

  for (int i=0; i<10; i+=2)
    v.push_back(i);

  int c = count_if(v.begin(), v.end(), [](int num) {
    return num % 2 == 0;
  });

  cout << c;

  return 0;
}
```

Choose one or more options:

1. ⬜ 9
2. ⬜ 5
3. ⬜ 4
4. ⬜ 3
5. ⬜ 8
6. ⬜ 6
7. ⬜ 7

Ex 6 (1.0 points)

Analyze the following code snippet. Indicate how many copy assignment operators and move assignment operators are called. Note that wrong answers imply a penalty in the final score.

```
class C {
private:
   ...
```

```
public:
   ...
};

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

Choose one or more options:

1. ⬜ 1 copy assignment(s) and 1 move assignment(s).
2. ⬜ 1 copy assignment(s) and 2 move assignment(s).
3. ⬜ 2 copy assignment(s) and 1 move assignment(s).
4. ⬜ 2 copy assignment(s) and 2 move assignment(s).
5. ⬜ 2 copy assignment(s) and 3 move assignment(s).
6. ⬜ 3 copy assignment(s) and 3 move assignment(s).
7. ⬜ 4 copy assignment(s) and 3 move assignment(s).

Ex 7 (1.0 point)

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

```
fd_set rset;
int fd1, fd2, maxfd;
...
maxfd = ((fd1>fd2) ? fd1 : fd2);
FD_ZERO (&rset);
FD_SET(fd1, &rset);
FD_SET(fd2, &rset);
select (maxfd, &rset, NULL, NULL, NULL);
if (FD_ISSET (fd1, &rset)) {
   read(fd1, &n, sizeof(int));
   ...
} else if (FD_ISSET (fd2, &rset)) {
   read(fd2, &n, sizeof(int));
   ...
}
```

Choose one or more options:

1. ⬜ The function select can be used only with reading descriptors.
2. ⬜ The function select returns the index of the descriptor ready.
3. ⬜ In the code, fd1, and fd2 are file descriptors on which we want to perform a reading operation.
4. ⬜ The line maxfd = ((fd1[0]>fd2[0]) ? fd1[0] : fd2[0]) is wrong because maxfd must be equal to the maximum descriptor plus 1.
5. ⬜ Is an example of asynchronous I/O in which we wait to read from multiple file descriptors.
6. ⬜ FD_ZERO, FD_SET and FD_ISSET are macros used to manipulate the data type fd_set.
7. ⬜ The code can be rewritten more efficiently using non-blocking I/O.

Ex 8 (1.5 points)

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

```
#define N 1024*1024
shmid = shmget(key, N, 0644 | IPC_CREAT);
r = shmat (shmid, NULL, 0);
i = 0;
j = N-1;
do {
   scanf ("%c", c1);
   r[i++] = c1;
    ...
   c2 = r[j--];
   fprintf (stdout, "%d\n", c2);
} while (...);
```

Choose one or more options:

1. ⬜ Function shmget is used by a process to attach the memory segment to its address space.
2. ⬜ At the end of the code segment, we should call function shmdt to detach the memory from the process.
3. ⬜ Before calling function shmget we must generate the key "key" using function ftok.
4. ⬜ Is an example of inter-process communication using a FIFO.
5. ⬜ Function shmat is used to obtain a shared memory identifier given the key.
6. ⬜ We should use read and write to manipulate the shared memory.
7. ⬜ If another process is also writing into the shared memory, we need a form of synchronization beyond the use of the memory between the processes.

Ex 9 (1.0 point)

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

Choose one or more options:

1. ⬜ The sequential container vectors are the C++ equivalent of C array and for that reason have a fixed size.
2. ⬜ The sequential containers `list` and `forward_list` are more efficient than `vectors` for random access.
3. ⬜ If `s` is a string, the operator `s.end()` denotes one past the last element.
4. ⬜ If s is a string, the operator `s.begin()` denotes one before the first element.
5. ⬜ The associative container `map` is always implemented using a hash-table.
6. ⬜ The associative container `set` stores pairs key-value.
7. ⬜ With containers the operations `c.insert()` and `c.emplace()` are equivalent.
8. ⬜ A multiset is a set in which each key may appear multiple times.
9. ⬜ The associative container `unordered_multiset` is implemented using a hash function.

Ex 10 (1.0 point)

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

Choose one or more options:

1. ⬜ There are two versions of the `new` operator, i.e., the normal one and the `nothrow` one.
2. ⬜ The C++ `new` operator is identical to the C `malloc` operator, and the two are interchangeable.

3. ⬜ Dangling pointers may cause memory leaks, but to avoid those it is sufficient to set them to `nullptr`.
4. ⬜ RAII is a programming technique binding a resource's life cycle to an object's life time.
5. ⬜ With RAII `new` and `delete` should never be used.
6. ⬜ A `weak_pointer` is a C++ pointer introduced to break circular dependency of `shared_pointers`.
7. ⬜ The construct `shared_ptr<list<int>> p;` define a shared `p` pointer to a list of integers.
8. ⬜ The construct `shared_ptr<list<int>> *p;` define a pointer to a list of integers.

Ex 11 (1.0 point)

Consider barrier and thread-pool constructs in C or C++. Indicate which of the following statements are correct. Note that more than one response can indeed be correct and that incorrect answers may imply a penalty on the final score.

Choose one or more options:

1. ⬜ The construct `pthread_barrier_wait` can be used to synchronize N thread only once, then the barrier must be re-initialized.
2. ⬜ The construct `pthread_barrier_wait` can be used only in acyclic situations.
3. ⬜ A barrier maintains multiple threads waiting for tasks to be allocated for concurrent execution.
4. ⬜ A thread pool maintains multiple threads waiting for tasks to be allocated for concurrent execution.
5. ⬜ A thread pool can be implemented by adopting a producer-and-consumer paradigm.
6. ⬜ A turnstile barrier is one in which each thread passing the barrier frees the next one.
7. ⬜ A turnstile barrier is a barrier in which all waiting threads are freed using a cycle that signals the waiting semaphore.
8. ⬜ A thread pool can be implemented using a thread throttle.
9. ⬜ A barrier can be implemented using a thread throttle.

Ex 12 (1.0 point)

Referring to C++ tasks with promises and futures, which of the following statements is correct? Note that incorrect answers may imply a penalty on the final score.

Choose one or more options:

1. ⬜ The primitive `std::asynch` runs a thread exactly as `std::thread`.
2. ⬜ The `std::thread` library offers a direct way to return a value to the caller.
3. ⬜ The policy launch::deferred indicates that a new thread is generated when its future is accessed.
4. ⬜ Function `future::get` is applied to obtain a valid future from a `std::thread`.
5. ⬜ A `std::future` is always associated with a `std::promise`.
6. ⬜ A promise is stored in the thread that generates it.
7. ⬜ The construct `auto future = promise.get_future()` associates a promise with a future.
8. ⬜ The construct `auto future = promise.get_future()` gets the promise and assigns it to the future.
9. ⬜ The template class `std::packaged_task<T>` wraps a function and allows it to produce a future with a return statement (is an RAII for futures).