CS69001 Computing Laboratory – I

Lab Test 2 **Date: 11–November–2019** **Maximum marks: 50**

1. How many lines are printed by the call `f(4)`, where `f` is defined as follows? Justify your answer. **(12)**

```
void f ( int n )
{
   int i;

   for (i=n-1; i>=1; --i) {
      printf("Hi\n");
      f(i);
      return;
   }
}
```

Solution Let $T(n)$ be the number of lines printed by $f(n)$ for $n \geqslant 1$. We have

$$T(1) = 0.$$

For $n \geqslant 2$, we have

$$T(n) = T(n-1) + T(n-2) + \cdots + T(1) + (n-1).$$

This gives

$$\begin{aligned} T(4) &= T(3) + T(2) + T(1) + 3 \\ &= [T(2) + T(1) + 2] + [T(1) + 1] + T(1) + 3 \\ &= T(2) + 6 \\ &= [T(1) + 1] + 6 \\ &= 7. \end{aligned}$$

There is a mistake in the question (or the solution). The given question has the (obvious) answer 3. The intended f was this.

```
void f ( int n )
{
   int i;

   for (i=n-1; i>=1; --i) {
      if (!fork()) {
         printf("Hi\n");
         f(i);
         return;
      }
   }
}
```

2. A game is played among three processes P_0, P_1, P_2. Initially, there are N coins in a bag. One of the three players starts the game. Then, the processes cyclically make moves. For example, if P_1 starts the game, then the moves are taken by $P_1, P_2, P_0, P_1, P_2, P_0, P_1, P_2, \ldots$ (in that sequence). In each move, a player takes t coins from the bag subject to two restrictions: (1) $1 \leqslant t \leqslant 5$, and (2) t cannot be the same as any one of the last three moves. Near the end, when no moves in the range $1 \leqslant t \leqslant 5$ are allowed (for example, only 2 coins are left, and the last three moves are $4, 2, 1$), the choice $t = 0$ is allowed. The game stops when all coins are taken from the bag. The player having collected the maximum number of coins wins.

The process you launch by running your code is called Q. This process forks the three child processes P_0, P_1, P_2 (the players). A shared array M of four integers is used to store the current state of the game. $M[0]$ stores the number of coins left, whereas $M[1], M[2], M[3]$ store the last three moves ($M[3]$ is the most recent move). In order to sequence the moves among the players, a semaphore array S with three semaphores is used. For $i = 0, 1, 2$, Player P_i waits on P_i and signals P_{i+1}, where $i + 1$ is treated modulo 3.

In the following parts, write code snippets for specific subtasks. You do not have to write the entire code. Assume that `P()` and `V()` operations are defined. You should declare all variables used in your snippets.

(a) Write the initialization code to be run by Q. This involves two tasks. First, Q creates and initializes the shared memory M. Assume that N is supplied by the user. So Q sets $M[0] = N$, and $M[1] = M[2] = M[3] = 0$. Q also creates an array S of three semaphores, and sets the value of each semaphore to zero. **(12)**

Solution

```
int N, shmid, semid, *M;

scanf("%d", &N);
shmid = shmget(IPC_PRIVATE, 4 * sizeof(int), 0644 | IPC_CREAT);
M = (int *)shmat(shmid, 0, 0);
M[0] = N; M[1] = M[2] = M[3] = 0;

semid = semget(IPC_PRIVATE, 3, 0644 | IPC_CREAT);
semctl(semid, 0, SETVAL, 0);
semctl(semid, 1, SETVAL, 0);
semctl(semid, 2, SETVAL, 0);
```

(b) The parent process Q forks the player processes P_0, P_1, P_2. Each player process calls a function **`playgame()`** immediately after it is forked (this function is to be implemented in Part (c)). Q chooses a random process $i \in \{0,1,2\}$, and signals the i-th semaphore in S, so that this P_i starts the game. Q then waits for all the player processes to exit. After this, Q removes the shared-memory segment and the semaphore array, and exits. Write this part of the code (starting from the forking of the player processes). **(12)**

Solution

```
int pno;

for (pno=0; pno<3; ++pno) {
   if (!fork()) playgame(pno,shmid,semid);
}

pno = rand() % 3;
V(semid, pno);

for (pno=0; pno<3; ++pno) wait(NULL);

shmdt(M); shmctl(shmid, IPC_RMID, 0);
semctl(semid, IPC_RMID, 0);

exit(0);
```

(c) Write the code for `playgame()` that each player process P_i runs. Clearly mention all the parameters you should pass to this function. Each P_i plays the game as follows. First, P_i waits on the i-th semaphore in S. When woken up, it enters a loop which runs as long as there are coins left in the bag. In the loop, P_i calls a function `bestmove()` to get the best allowed move t at this point (t may be zero near the end of the game). You do not have to write the function `bestmove()`; just call it with a mention of what parameter(s) you need to pass to it. P_i then makes its next move by taking away t coins from the bag. P_i should appropriately update the shared memory M to reflect this move. Finally, P wakes up the cyclically next process, and again goes to wait on the i-th semaphore. P_i should also keep track of the total sum of coins taken by it in all its moves. After the loop breaks, it prints that sum, and exits. **(14)**

Solution

```
void playgame ( int pno, int shmid, int semid )
{
   int *M, mysum, npno, t, i;

   M = shmat(shmid,0,0);
   npno = pno + 1; if (npno == 3) npno = 0;
   mysum = 0;
   P(semid,pno);
   while (M[0]) {
      t = bestmove(M);
      M[0] -= t;
      M[1] = M[2];
      M[2] = M[3];
      M[3] = t;
      mysum += t;
      V(semid,npno);
      P(semid,pno);
   }
   V(semid,npno);
   printf("Player %d: Total = %d\n", pno, mysum);
   shmdt(M);
   exit(0);
}
```