Project 4 IPC
A Simple Game with Shared Memory

Project Descriptions

This project aims to make students familiar with C/C++ programming on inter-process communication (IPC) with shared memory. It requires programming skills on process creation, termination, inter-process communications using shared memory, and resource control (creation and destroy of shared memory). You are required to implement a small multi-process program that simulates a simple game.

To set up the game, initially a base number (*base*) is assigned to 5, and then all players form a ring to play. The first player picks a random number (***rand_num***) between 1 and 10, and compares it against *base*. If ***rand_num*** is equal to or larger than *base*, then the player calls the next player for the next play and stays in the ring. If ***rand_num*** is smaller, then the player calls the next player but leaves the ring. In either case, the player always updates the *base* with its own ***rand_num***. The game continues until only one player is left in the ring, and then the last player becomes the winner and the game ends.

Specification and Requirement

1. The initial *base* is set to 5; it should be stored in shared memory.
2. The parent process creates child processes based on the input, ***num_process***. The position of each process in the ring is determined in the order of its creation, i.e. the parent process is player #1 (player_id = 1), the first child process is player #2 (player_id = 2), and so on. The processes use a shared memory segment to communicate with each other.
3. The parent process is the first player. In each play, a player picks a random number (0 <= *rand_num* <*10*) and compares it against the base value (*base*). If *rand_num* is less than *base*, then the player updates *base* with *rand_num*, prints a message (the message format is shown in the next section), calls the next player and exits itself out of the game. The next player in the ring continues. If *rand_num* is equal to or larger than *base*, the player does the same thing except that it does not quit and stay in the game for next turns.
4. When there is only one player (process) left in the ring, the player prints out a message "winner" and the total number of plays, and then exits the program.
5. Use *printf()* for printing out. See the provided base code for the format.
6. If the program executable is named *proj4*. Command line format should be,
   ./proj4 num_process rand_num_seed
   where *num_process* is the number of processes that will participate in the game including parent (max number is 20 for simplicity, but I wish everyone's program could remove this restriction), and *rand_num_seed* is the seed to generate deterministic pseudo-random integer numbers.

7. To generate a deterministic sequence random numbers, you **must** use srand (*player_id* + *seed*) before rand(). See the following example:

```
void play_game ( int player_id ) { // Method for all player_id = [1..n]
  …
  srand(player_id + rand_num_seed); // rand_num_seed is from command line argument
  …
  while (…) {
     rand_num = 1 + rand() % 10; // Get a random number between 1 - 10
    …
  }
  …
}
```

8. The parent process **terminates** only after the game ends and all its child processes terminate, although it **leaves** the game according to the game rules.

Input/Output with Demo Program

Example: The example program makes a 10-player game with random number seed 3. **Note**: Running it in Linux will have a different result due to a different pseudo-random number sequence.

```
cs1 > ./proj4
Usage: proj4 num_process rand_num_seed
cs1 > ./proj4 10 3
Game started! Total 10 players, parent pid is 29917, base is 5
In : player 1 (pid 29917) stays (base 5, rand_num 7)
Out: player 2 (pid 29918) is leaving (base 7, rand_num 5)
In : player 3 (pid 29919) stays (base 5, rand_num 6)
Out: player 4 (pid 29920) is leaving (base 6, rand_num 4)
In : player 5 (pid 29921) stays (base 4, rand_num 4)
Out: player 6 (pid 29922) is leaving (base 4, rand_num 2)
In : player 7 (pid 29923) stays (base 2, rand_num 3)
Out: player 8 (pid 29924) is leaving (base 3, rand_num 1)
In : player 9 (pid 29925) stays (base 1, rand_num 1)
Out: player 10 (pid 29926) is leaving (base 1, rand_num 0)
In : player 1 (pid 29917) stays (base 0, rand_num 6)
Out: player 3 (pid 29919) is leaving (base 6, rand_num 5)
Out: player 5 (pid 29921) is leaving (base 5, rand_num 4)
In : player 7 (pid 29923) stays (base 4, rand_num 4)
Out: player 9 (pid 29925) is leaving (base 4, rand_num 3)
Out: player 1 (pid 29917) is leaving (base 3, rand_num 1)
WINNER is process 7 (pid 29923), total 16 times play
```

Hints

• Input parameters are received through command line arguments: argc, argv
• System calls and library functions: printf, atoi, fork, getpid, getppid, wait, shmget,shmat, shmctl, srand and rand.
• Access to shared memory can be controlled **without** any mutex or semaphore. In other words, we do not need to use any synchronization tools like mutex locks or semaphores to do synchronization in this project. Think about it and why?
• Sample shared memory data structure is shown below.

```
struct my_shared_mem {
  int num_proc; /* Total # of processes, including the parent */
  int num_running; /* # of processes which are currently in the game */
  int next; /* Next player id, player_id = [1..num_proc] */
  int base_num; /* Base number to compare (initial value is 5) */
  bool in_out[MAX_PROC];/* flag, true: in the game, false: out the game */
  int count; /* total count, how long have we been playing */
};
```

• A document about shared memory is posted online in Section of "Suppl. Materials"

Submission:

1. All source code files
2. A readme file that briefly describes each file, and how to run the program
3. A Makefile
4. Use **tar** to pack all files above into a package named **project4.tar**
5. Due date: **11/30/2009, Monday, 1:30PM**
6. Submission command:

/home/fac/zhuy/fall09_340/SubmitHW340 p4 project4.tar

Grading Policy:

1. If you do not have the files specified above, it will result in a score of zero.
2. If the program cannot be compiled and run, it will result in a score of zero.
3. Collaboration will result in a score of zero for all the students involved.