Yeukhon Wong 8066
Fall 2012

Dad-Son Problem

To solve the problem, we have to create two semaphores.

```
/* STUDENT SOLUTION BEGIN */
int mutex, roomToAccess;
mutex=semget(IPC_PRIVATE,1,PERMS | IPC_CREAT);
roomToAccess=semget(IPC_PRIVATE,1,PERMS | IPC_CREAT);
sem_create(mutex, 1);
sem_create(roomToAccess, N_Att);
/* STUDENT SOLUTION END */
```

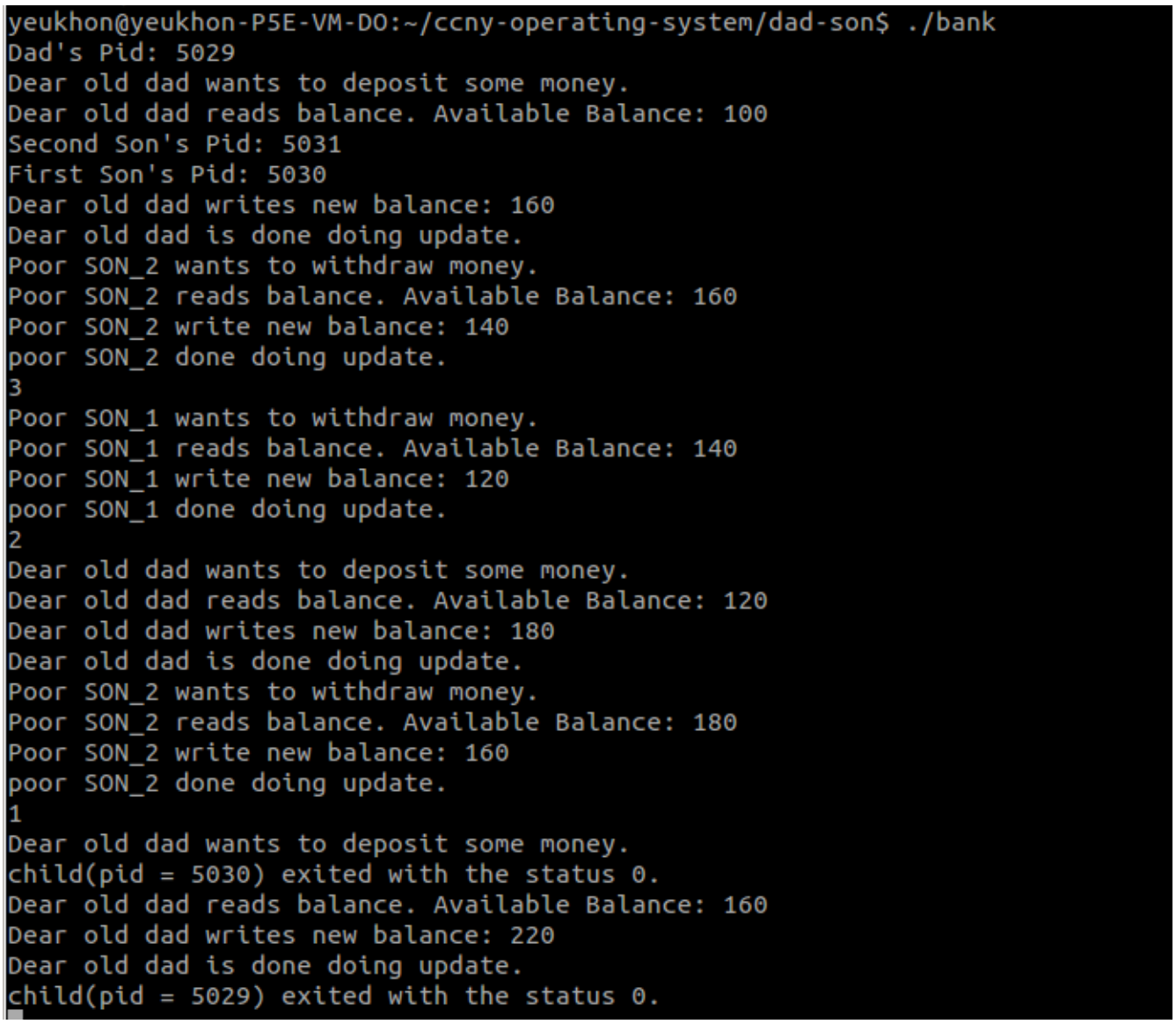
Then we perform the following before entering critical section and before leaving critical section.

```
P(roomToAccess);
P(mutex);
  { CS }
V(mutex);
V(roomToAccess);
```

This works because Dad / Son1 / Son2 must wait until there is room to access and mutex is positive again. The screenshot below shows when N = 1 (Dad deposits once). Dad is producer, and sons are consumers.

```
yeukhon@yeukhon-P5E-VM-DO:~/ccny-operating-system/dad-son$ ./bank
Dad's Pid: 5009
Dear old dad wants to deposit some money.
Dear old dad reads balance. Available Balance: 100
Second Son's Pid: 5011
First Son's Pid: 5010
Dear old dad writes new balance: 160
Dear old dad is done doing update.
child(pid = 5009) exited with the status 0.
Poor SON_2 wants to withdraw money.
Poor SON_2 reads balance. Available Balance: 160
Poor SON_2 write new balance: 140
poor SON_2 done doing update.
3
Poor SON_1 wants to withdraw money.
Poor SON_1 reads balance. Available Balance: 140
Poor SON_1 write new balance: 120
poor SON_1 done doing update.
2
Poor SON_2 wants to withdraw money.
Poor SON_2 reads balance. Available Balance: 120
Poor SON_2 write new balance: 100
poor SON_2 done doing update.
1
child(pid = 5010) exited with the status 0.
```

Here is the screenshot when N = 3 (dad deposits three times, but keeps max attempt = 3 for sons).



```c
#include <stdio.h>
#include <stdlib.h>
#include "sem.h"

#define CHILD                   0               /* Return value of child proc from
fork call */
#define TRUE                    0
#define FALSE                   1
#define PERMS                 0666
FILE *fp1, *fp2, *fp3, *fp4;            /* File Pointers */

main()
{
  int pid;                          // Process ID after fork call
  int i;                            // Loop index
  int N;                            // Number of times dad does update
  int N_Att = 3;                    // Number of time sons allowed to do update
  int status;                       // Exit status of child process
  int bal1, bal2;                   // Balance read by processes
  int flag, flag1;                  // End of loop variables

  /* STUDENT SOLUTION BEGIN */
  int mutex, roomToAccess;
  mutex=semget(IPC_PRIVATE,1,PERMS | IPC_CREAT);
  roomToAccess=semget(IPC_PRIVATE,1,PERMS | IPC_CREAT);
  sem_create(mutex, 1);
  sem_create(roomToAccess, N_Att);
  /* STUDENT SOLUTION END */


  //Initialize the file balance to be $100
  fp1 = fopen("balance","w");
  bal1 = 100;
  fprintf(fp1, "%d\n", bal1);
  fclose(fp1);

  //Initialize the number of attempts to be 20
  fp4 = fopen("attempt", "w");
  N_Att = 3;
  fprintf(fp4, "%d\n", N_Att);
  fclose(fp4);

  //Create child processes that will do the updates
  if ((pid = fork()) == -1)
    {
      //fork failed!
      perror("fork");
      exit(1);
    }

  if (pid == CHILD)
    {
      //First Child Process. Dear old dad tries to do some updates.
      printf("Dad's Pid: %d\n",getpid());
      N=1;
      for(i=1; i<=N; i++)
        {
          /* STUDENT SOLUTION BEGIN */
```

```
        P(roomToAccess);
        P(mutex);
        /* STUDENT SOLUTION END */

        printf("Dear old dad wants to deposit some money.\n");
        fp1 = fopen("balance", "r+");
        fscanf(fp1, "%d", &bal2);
        printf("Dear old dad reads balance. Available Balance: %d \n", bal2);

        //Dad has to think (0-14 sec) if his SON is really worth it
        sleep(rand()%2);
        fseek(fp1,0L,0);
        bal2 += 60;
        printf("Dear old dad writes new balance: %d \n", bal2);
        fprintf(fp1, "%d \n", bal2);
        fclose(fp1);

        printf("Dear old dad is done doing update. \n");
        sleep(rand()%2);    /* Go have coffee for 0-4 sec. */

        /* STUDENT SOLUTION BEGIN */
        V(mutex);
        V(roomToAccess);
        /* STUDENT SOLUTION END */

      }
  }

else
  {
    //Parent Process. Fork off another child process.
    if ((pid = fork()) == -1)
      {
        //Fork failed!
        perror("fork");
        exit(1);
      }
    if (pid == CHILD)
      {
        printf("First Son's Pid: %d\n",getpid());
        //Second child process. First poor SON tries to do updates.
        flag = FALSE;
        while(flag == FALSE)
          {

            /* STUDENT SOLUTION BEGIN */
            P(roomToAccess);
            P(mutex);
            /* STUDENT SOLUTION END */

            fp3 = fopen("attempt" , "r+");
            fscanf(fp3, "%d", &N_Att);
            if(N_Att == 0)
              {
                fclose(fp3);
                flag = TRUE;
              }
            else
```

```
                {

                  printf("Poor SON_1 wants to withdraw money.\n");
                  fp2 = fopen("balance", "r+");
                  fscanf(fp2,"%d", &bal2);
                  printf("Poor SON_1 reads balance. Available Balance: %d \n",
bal2);
                  if (bal2 == 0)
                    {
                      fclose(fp2);
                      fclose(fp3);
                    }
                  else
                    {
                      sleep(rand()%5);
                      fseek(fp2,0L, 0);
                      bal2 -=20;
                      printf("Poor SON_1 write new balance: %d \n", bal2);
                      fprintf(fp2,"%d\n", bal2);
                      fclose(fp2);
                      printf("poor SON_1 done doing update.\n");
                      printf("%d\n", N_Att);
                      fseek(fp3,0L, 0);
                      N_Att -=1;
                      fprintf(fp3, "%d\n", N_Att);
                      fclose(fp3);
                    }
                }

              /* STUDENT SOLUTION BEGIN */
              V(mutex);
              V(roomToAccess);
              /* STUDENT SOLUTION END */
            }

        }
      else
        {
          //Parent Process. Fork off one more child process.
          if ((pid = fork()) == -1)
            {
              //fork failed!
              perror("fork");
              exit(1);
            }
          if (pid == CHILD)
            {
              printf("Second Son's Pid: %d\n",getpid());
              //Third child process. Second poor SON tries to do updates.
              flag1 = FALSE;
              while(flag1 == FALSE)
                {
                  /* STUDENT SOLUTION BEGIN */
                  P(roomToAccess);
                  P(mutex);
                  /* STUDENT SOLUTION END */

                  fp3 = fopen("attempt" , "r+");
```

```
                    fscanf(fp3, "%d", &N_Att);
                    if(N_Att == 0)
                      {
                        fclose(fp3);
                        flag = TRUE;
                      }
                    else
                      {
                        printf("Poor SON_2 wants to withdraw money.\n");
                        fp2 = fopen("balance", "r+");
                        fscanf(fp2,"%d", &bal2);
                        printf("Poor SON_2 reads balance. Available Balance: %d \n",
bal2);
                        if (bal2 == 0)
                          {
                            fclose(fp2);
                            fclose(fp3);
                          }
                        else
                          {
                            sleep(rand()%2);
                            fseek(fp2,0L, 0);
                            bal2 -=20;
                            printf("Poor SON_2 write new balance: %d \n", bal2);
                            fprintf(fp2,"%d\n", bal2);
                            fclose(fp2);

                            printf("poor SON_2 done doing update.\n");
                            printf("%d\n", N_Att);
                            fseek(fp3,0L, 0);
                            N_Att -=1;
                            fprintf(fp3, "%d\n", N_Att);
                            fclose(fp3);
                          }
                      }
                    /* STUDENT SOLUTION BEGIN */
                    V(mutex);
                    V(roomToAccess);
                    /* STUDENT SOLUTION END */
                  }
              }
            else
              {
                //Now parent process waits for the child processes to finish
                pid = wait(&status);
                printf("child(pid = %d) exited with the status %d. \n", pid,
status);

                pid = wait(&status);
                printf("child(pid = %d) exited with the status %d. \n", pid,
status);

                pid = wait(&status);
                printf("child(pid = %d) exited with the status %d. \n", pid,
status);
              }
            exit(0);
          }
```

```
      exit(0);
    }
  exit(0);
}
```