CSE 380: Homework 2: Synchronization

Due : Thursday, October 2, 2003

Submit a hardcopy solution of the problems in class on Oct 2, and submit code and documentation for the programs using the `turnin` command to the account `cse380@eniac` by 11.59pm on Oct 2. The code should include comments which adequately describe the solution, as described below.

Reminder: Copying the solution from a fellow student, a website, or some other source, is a violation of the University policies on academic integrity.

Implementi

Recall the dining p [illegible] le eating spaghetti periodically. In t [illegible] system calls for semaphores. You [illegible] roblem:

a) a solution to [illegible] red memory, and semaphores

b) a solution usi [illegible]

A skeleton for [illegible]

`http://www.cis.` [illegible]

It specifies the bas [illegible] ncludes a makefile that will allow you [illegible] `hread` to generate the program for p [illegible] the targets in the makefile.

This assignme [illegible] ctions, and `POSIX` threads. You are [illegible]

As far as the solution to the problem is concerned, you can use class notes, textbook, or design your own solution.

Part One: Using Process and Shared Memory

For part one, your solution should work in the following way (see `dinPhil_proc.c` in the skeleton):

1. Create one process for each philosopher

2. The number of philosophers should be the first argument to your program. The input range is restricted from three to 20 philosophers, inclusive.

3. Each philosopher should eat *bites* times, which is the second input parameter to the program. Count the number of times each philosopher eats, and when (s)he has eaten the maximum number of bites, (s)he stops eating (the corresponding process exits).

4. For mutual exclusion, use `POSIX` semaphores. Relevant system calls are: `sem_init()`, `sem_post()`, `sem_wait`, `sem_getvalue` and `sem_destroy()`.

   The semaphores should be allocated in shared memory (see Appendix A on the system support for doing this in Solaris).

5. Simulate thinking and eating using the `random` and `sleep` system calls.

6. Your code should be well documented with sufficient explanation of high-level objectives as well as of details. Your documentation will serve as the main justification for the correctness of your program. It is your responsibility to convince the graders that your solution makes sense.

Part Two: Usi

Threads should be … An example of a creating pthreads … requirements for part one with the …

1. Create one t… n.

2. Shared mem…

3. Mutual exclu… either semaphores as in part o… x_unlock.

Submission

We have created … es, your program, the output, and th… m the makefile, or alter the names of … sections, one for each part of the a… your name, email address, and a des… of each of the two sections. Please i… solution.

- use turnin with the following syntax:

  ```
  turnin -ccse380 -phw2 Makefile dinPhil_thread.c dinPhil_proc.c
  ```

- For every critical section in your code, add statements that identify the entering and exiting from that section. If your program has two critical sections, then the lock of the mutex before the first should be directly followed by the line:

  ```
  printf("Starting critical section 1\n");
  ```

- Unlocking the mutex to exit the critical section should be directly preceded by this line:

  ```
  printf("Stopping critical section 1\n");
  ```

Use the exact format above.

- When philosopher 'i' takes their forks, print a line with the following format to stdout:

```
printf("Philosopher %d taking forks\n", i);
```

Similarly, when philosopher 'i' puts their forks down, print a line with the following format:

```
printf("Philosopher %d putting forks\n", i);
```

- Keep track of each time a philosopher eats. When philosopher 'i' eats for the 'j'-th time, print t

```
pri j);
```

Use the man p tion on necessary system calls (e.g.

Appendix A: Shared Memory System Calls

This section gives information on how Unix processes can request and use shared memory segments. For every shared memory segment, the kernel maintains the following structure of information:

```
/*
 *      There is a shared mem id data structure (shmid_ds) for each
 *      segment in the system.
 */
struct shmid_ds {
        struct ipc_perm shm_perm;       /* operation permission struct */
        size_t                                              bytes */
       /*.... s
        pid_t                                              */
        pid_t
        shmatt_                                            */
        ulong_t                                            */
        time_t
        time_t
        time_t                                             */
};
```

The ipc_perm   memory segment.

```
struct ipc_perm
        uid_t
        gid_t
        uid_t
        gid_t
        mode_t
        uint_t
        key_t   key;    /* key */
};
int shmget(key_t key, int size, int shmflag);
```

The *shmflag* argument specifies the low-order 9 bits of the *mode* for the shared memory, and whether a new segment is being created or if an existing one is being referenced.

The *shmflag* argument is a combination of the constants:

Numeric	Symbolic	Description
0400	SHM_R	Read by owner
0200	SHM_W	Write by owner
0040	SHM_R>>3	Read by group
0020	SHM_W>>3	Write by group
0004	SHM_R>>6	Read by world
0002	SHM_W>>6	Write by world
	IPC_CREAT	See below
	IPC_EXCL	See below

The rules for whether a new shared memory segment is created or whether an existing one is referenced are:

- Specifying a [illegible] hannel is created.
- Setting the I [illegible] y for the specified *key*, if it doe [illegible] returned.
- Setting both [illegible] ord creates a new entry of the [illegible] an existing entry is found, an [illegible]

Upon creation [illegible] emory identifier is initialized as follo [illegible]

- The values [illegible] shm_perm.gid are set equal to [illegible] ely, of the calling process.
- The access [illegible] access permission bits of shmf [illegible]
- The values [illegible] set equal to 0.
- The shm_ctime is set equal to the current time.

The following example illustrates how shared memory can be used.

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>

int main (int argc, char *argv[]){
 int shmid; /* shared memory ID */
 char *p;   /* pointer to shared memory area */

/* reserve a 10-byte physical memory segment
   using the pid as the key */
```

```
   shmid = shmget ((key_t) getpid (), 10, 0666|IPC_CREAT);
   if (shmid == -1) {
     puts ("shmget failed");
     exit (1);
   }

/* attach the shared memory for use by this program */
  p = (char *) shmat (shmid, (char *) 0, 0);
  strcpy (p, "hello"); /* put string into shared memory */
  puts (p);

/* detach the s
  shmdt (p);

/* remove the s
  shmctl (shmid
}
```