System and Device Programming

Review Exercises

Stefano Quer
Dipartimento di Automatica e Informatica
Politecnico di Torino

Exercise

- A file, of undefined length and in ASCII format, contains a list of integer numbers
- Write a program that, after receiving a value k (integer) and a string from command line, generates k threads and wait them
- Each thread
  - Reads concurrently the file, and performs the sum of the read integer numbers
  - When the end of file is reached, it must print the number of integer numbers it has read and the computed sum
  - Terminates

Exercise

- After all threads terminate, the main thread has to print the total number of integer numbers and the total sum
- Example

File format: file.txt

```
7
9
2
-4
15
0
3
```

Example of execution

```
> pgrm 2 file.txt
Thread 1: Sum=18 #Line=3
Thread 2: Sum=14 #Line=4
Total   : Sum=32 #Line=7
```

Solution

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <unistd.h>
#include <sys/types.h>
#include <semaphore.h>
#include <pthread.h>

#define L 100
struct threadData {
  pthread_t threadId;
  int id;
  FILE *fp;
  int line;
  int sum;
};
static void *readFile (void *);
sem_t sem;
```

Includes, variables and prototypes

It must be unique (i.e., it is global or it is passed as parameter to threads through this structure)

Solution

Main
Part 1

```
int main (int argc,char *argv[]) {
  int i, nT, total, line;
  struct threadData *td;
  void *retval;
  FILE *fp;

  nT = atoi (argv[1]);
  td = (struct threadData *) malloc
       (nT * sizeof (struct threadData));
  fp = fopen (argv[2], "r");
  if (td==NULL || fp==NULL) {
    fprintf (stderr, "Error ...\n");
    exit (1);
  }
  sem_init (&sem, 0, 1);
```

Not shared

Init to 1

Solution

Main Part 2

File pointer common to all the threads

```
  for (i=0; i<nT; i++) {
    td[i].id = i;
    td[i].fp = fp;
    td[i].line = td[i].sum = 0;
    pthread_create (&(td[i].threadId),
      NULL, readFile, (void *) &td[i]);
  }
  total = line = 0;
  for (i=0; i<nT; i++) {
    pthread_join (td[i].threadId, &retval);
    total += td[i].sum;
    line += td[i].line;
  }
  fprintf (stdout, "Total: Sum=%d #Line=%d\n",
    total, line);
  sem_destroy (&sem);
  fclose (fp);
  return (1);
}
```

Solution

Thread function

```
static void *readFile (void *arg){
  int n, retVal;
  struct threadData *td;
  td = (struct threadData *) arg;
  do {
    sem_wait (&sem);
    retVal = fscanf (td->fp, "%d", &n);
    sem_post (&sem);
    if (retVal!=EOF) {
      td->line++;
      td->sum += n;
    }
    sleep (1);  // Delay Threads
  } while (retVal!=EOF);
  fprintf (stdout, "Thread: %d Sum=%d #Line=%d\n",
    td->id, td->sum, td->line);
  pthread_exit ((void *) 1);
}
```

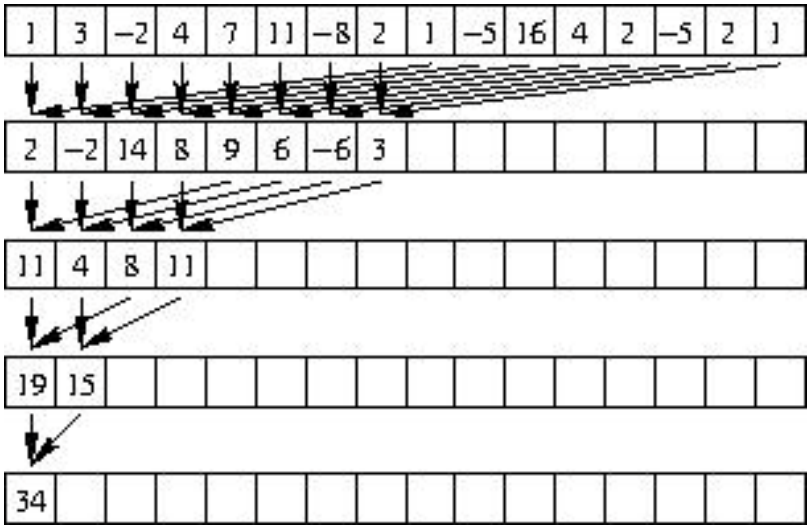
Mutual exclusion for file reading

Exercise

- Given an array vet and its size n (which is supposed to be equal to a power of 2), write the function

```
int array_sum (int *vet, int n);
```

- Which computes the sum of the elements of the array as represented in the picture



Exercise

- In particular
  - All sums must be executed in parallel by n/2 separate threads
  - Each thread is associated with one of the first n/2 cells of the array
- Note that the number of sums each thread will have to execute depends on the position of the cell
- Manage synchronization between threads with semaphores, so that all sums are made respecting precedence

Solution

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
#include <semaphore.h>

typedef struct {
  int *vet;
  sem_t *sem;
  int n;
  int id;
} args_t;

... main ...
```

Array of n elements

Array of n/2 semaphores

User thread identifier

Solution

```
int array_sum (int *vet, int n) {
  int k=n/2; pthread_t *tids; args_t *args; sem_t *sem;
  tids = (pthread_t *) malloc (k*sizeof(pthread_t));
  sem = (sem_t *) malloc (k*sizeof(sem_t));
  for (int i=0; i<k; ++i) sem_init(&sem[i], 0, 0);
  args = (args_t *) malloc (k*sizeof(args_t));
  for (int i=0; i<k; ++i) {
    args[i].id = i; args[i].vet = vet;
    args[i].n = n; args[i].sem = sem;
  }
  for (int i=0; i<k; ++i)
    pthread_create (&tids[i], NULL, adder, &args[i]);
  pthread_join (tids[0], NULL);
  for (int i=0; i<k; ++i) sem_destroy(&sem[i]);
  free (tids);
  free (sem);
  free (args);
  return vet[0];
}
```

n/2 Ts and Sems

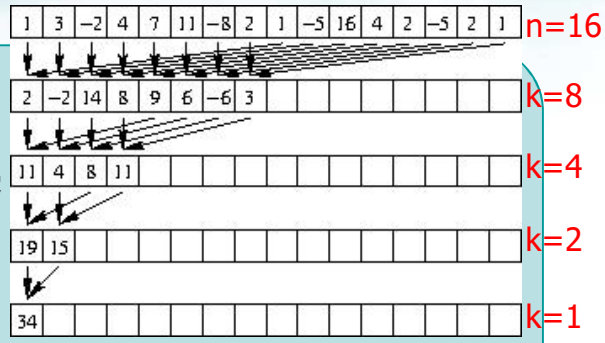
Initialize

Run threads

Wait for threads and free memory

Solution

```
void *adder (void * arg) {
  sem_t *sem = ((args_t *) arg)->sem;
  int *vet = ((args_t *) arg)->vet;
  int id = ((args_t *) arg)->id;
  int n = ((args_t *) arg)->n;
  int k = n/2;
  while (k != 0) {
    if (k < n/2)
      sem_wait (&sem[id + k]);
    vet[id] += vet[id + k];
    k = k/2;
    if (id >= k) {
      sem_post (&sem[id]);
      break;
    }
  }
  pthread_exit(0);
}
```



k = # iterations

Wait for the previous sum to be done
id ∈ [0, n/2[

Make the sum

My sum has been done

This thread must stop