|  |
| --- |
| #include <sys/types.h> |
|  | #include <sys/wait.h> |
|  | #include <sys/sem.h> |
|  | #include <stdbool.h> |
|  | #include <sys/shm.h> |
|  | #include <unistd.h> |
|  | #include <stdio.h> |
|  | #include <stdlib.h> |
|  | #include <string.h> |
|  | #include "shm.h" |
|  |  |
|  | #define SEMKEY 203948 |
|  | #define SHMKEY 102985 |
|  | #define NUMSEMS 5 // # of semaphores in semaphore set (one for each process) |
|  |  |
|  | /\*\* |
|  | \* Function to initialize values of all the semaphores in the semaphore set |
|  | \*/ |
|  | static int set\_semvalue(int semid, int semnum) |
|  | { |
|  | union semun sem\_union; |
|  | sem\_union.val = 1; |
|  | if (semctl(semid, semnum, SETVAL, sem\_union) == -1) |
|  | return 0; |
|  | return 1; |
|  | } |
|  |  |
|  | /\*\* |
|  | \* Deletes all semaphores in semaphore set |
|  | \*/ |
|  | static void del\_semvalue(int semid) |
|  | { |
|  | union semun sem\_union; |
|  | if (semctl(semid, 0, IPC\_RMID, sem\_union) == -1) |
|  | fprintf(stderr, "Failed to delete semaphore\n"); |
|  | } |
|  |  |
|  | /\*\* |
|  | \* Wait function |
|  | \*/ |
|  | static int p(int semid, int child) |
|  | { |
|  | struct sembuf sem\_b; |
|  | sem\_b.sem\_num = child; |
|  | sem\_b.sem\_op = -1; /\* P() sem\_b.sem\_flg = SEM\_UNDO; if (semop(sem\_id, &sem\_b,\*/ |
|  | if (semop(semid, &sem\_b, 1) == -1) |
|  | { |
|  | fprintf(stderr, "semaphore\_p failed\n"); |
|  | return 0; |
|  | } |
|  | return 1; |
|  | } |
|  |  |
|  | /\*\* |
|  | \* Signal function |
|  | \*/ |
|  | static int v(int semid, int child) |
|  | { |
|  | struct sembuf sem\_b; |
|  | sem\_b.sem\_num = child; |
|  | sem\_b.sem\_op = 1; /\* V() \*/ |
|  | sem\_b.sem\_flg = SEM\_UNDO; |
|  | if (semop(semid, &sem\_b, 1) == -1) |
|  | { |
|  | fprintf(stderr, "semaphore\_v failed\n"); |
|  | return 0; |
|  | } |
|  | return 1; |
|  | } |
|  |  |
|  | /\*\* |
|  | \* Creates and array from user inputted integers |
|  | \*/ |
|  | void collectData(void) |
|  | { |
|  | printf("Please enter 5 integers:\n"); |
|  | for (int i = 0; i < SIZE; i++) |
|  | { |
|  | scanf("%d", &data[i]); |
|  | } |
|  | } |
|  |  |
|  | /\*\* |
|  | \* Prints the array |
|  | \*/ |
|  | void printData(void) |
|  | { |
|  | for (i = 0 to SIZE of array) |
|  | { |
|  | Print "data[i]” |
|  | } |
|  | Print "\n(next line)" |
|  | } |
|  |  |
|  | /\*\* |
|  | \* Checks if array is sorted in decreasing order |
|  | \*/ |
|  | bool sorted(void) |
|  | { |
|  | for (i <- 0 to SIZE of array) |
|  | { |
|  | if (data at [i] < data at [i + 1]) |
|  | { |
|  | return false; |
|  | } |
|  | } |
|  | return true; |
|  | } |
|  |  |
|  | /\*\* |
|  | \* Sorts the array |
|  | \*/ |
|  | void sort(int semid, int pi, char debug) |
|  | { |
|  | int tmp; |
|  | while (data not sorted) |
|  | { |
|  | if (data at [pi + 1] > data at [pi]) |
|  | { |
|  | Request keys for two adjacent array elements |
|  |  |
|  |
|  |
|  |
|  | Swap elements |
|  |  |
|  | Release keys for two adjacent array elements |
|  |  |
|  |  |
|  |
|  |
|  |
|  |
|  |
|  |
|  | if (debug == YES) |
|  | Print "Performed swapping”  Increment Pi |
|  | } |
|  | else |
|  | { |
|  | if (debug == YES) |
|  | Print "No swapping" |
|  | } |
|  | } |
|  | } |
|  |  |
|  | /\*\* |
|  | \* Gets median if sorted |
|  | \*/ |
|  | int getMedian(void) |
|  | { |
|  | if (data sorted) |
|  | { |
|  | return data at [2]; |
|  | } |
|  | else return 0; |
|  | } |
|  |  |
|  | /\*\* |
|  | \* Gets maximum if sorted |
|  | \*/ |
|  | int getMax(void) |
|  | { |
|  | if (data sorted) |
|  | { |
|  | return data at [0]; |
|  | } |
|  | else return 0; |
|  | } |
|  |  |
|  | /\*\* |
|  | \* Gets minimum if sorted |
|  | \*/ |
|  | int getMin(void) |
|  | { |
|  | if (data sorted) |
|  | { |
|  | return data at [4]; |
|  | } |
|  | else return 0; |
|  | } |
|  |  |
|  | int main(void) |
|  | { |
|  | key\_t semkey, shmkey; |
|  | int semid, shmid; // semaphore set and shared memory identities, respectively |
|  | void \*shared\_mem; |
|  | char debug; // for debug mode |
|  |  |
|  | semid <- semget(SEMKEY, NUMSEMS, 0666 | IPC\_CREAT); // creating semaphore set of 5 |
|  | if (semid == -1) |
|  | { |
|  | Print "main: semget() failed\n" |
|  | return -1; |
|  | } |
|  |  |
|  | for (int i <- 0 to NUMSEMS) // Initializes semaphores |
|  | { |
|  | if (set\_semvalue(semid, i) not true) |
|  | { |
|  | Print "Failed to initialize semaphore\n" |
|  | exit(EXIT\_FAILURE); |
|  | } |
|  | } |
|  |  |
|  | shmid <- shmget(SHMKEY, SIZE \* sizeof(int), 0666 | IPC\_CREAT); // shared memory identity |
|  | if (shmid == -1) |
|  | { |
|  | print "shmget failed" |
|  | exit(EXIT\_FAILURE); |
|  | } |
|  |  |
|  | shared\_mem <- shmat(shmid, (void \*)0, 0); // attaching shared memory to address space of process |
|  | if (shared\_mem == (void \*)-1) |
|  | { |
|  | print "shmat failed" |
|  | exit(EXIT\_FAILURE); |
|  | } |
|  |  |
|  | data <- (int \*)shared\_mem; // assigning shared memory to the struct in the header file "shm.h" |
|  |  |
|  | collectData(); |
|  | Print "Unsorted array: " |
|  | printData(); |
|  | Print "Debug Mode? [y/n]" |
|  | scanf(" %c", &debug); // input for debug mode |
|  |  |
|  | int a <- SIZE - 1; // number of child processes |
|  | pid\_t pids[a]; |
|  |  |
|  | /\* Start children. \*/ |
|  | for (int i <- 0 to a) |
|  | { |
|  | if (fork() fails) |
|  | { |
|  | perror("fork"); |
|  | abort(); |
|  | } |
|  | else if (process is a child) |
|  | { |
|  | sort(semid, i, debug); |
|  | exit(0); |
|  | } |
|  | } |
|  |  |
|  | while (array not sorted) |
|  | { |
|  | Check if array is sorted |
|  | } |
|  |  |
|  | /\* Wait for children to exit. \*/ |
|  | int status; |
|  | pid\_t pid; |
|  | while (a > 0) |
|  | { |
|  | pid <- wait(&status); |
|  | --a; // Remove pid from the pids array. |
|  | } |
|  |  |
|  | Print "Sorted array: " |
|  | printData(); |
|  | printf("Maximum: %d\n", getMax()); |
|  | printf("Minimum: %d\n", getMin()); |
|  | printf("Median: %d\n", getMedian()); |
|  |  |
|  | del\_semvalue(semid); // Delete all semaphores |
|  |  |
|  | if (shmdt(shared\_mem) == -1) |
|  | { |
|  | fprintf(stderr, "shmdt failed\n"); |
|  | exit(EXIT\_FAILURE); |
|  | } |
|  |  |
|  | if (shmctl(shmid, IPC\_RMID, 0) == -1) |
|  | { |
|  | fprintf(stderr, "shmctl failed\n"); |
|  | exit(EXIT\_FAILURE); |
|  | } |
|  | } |