Marco Duarte
CSCE 5640
October 30, 2017

Project #1

Overview

The goal of this project is to implement a semaphore-based solution to the 3-list problem. Using shared memory, a solution that is deadlock-free and is maximally concurrent was able to be implemented using seven semaphores.

List Implementation

Originally, an implementation using an actual list structure was attempted but led to many out of bounds errors. Realizing that allocating shared memory is just allocating a block of a certain size the implementation was changed to use array indexing which made the list implementation much easier. The size of the list is given by the first given parameter, argv[1]. If an N size list is requested an N+1 block is put into shared memory, with the last block used to hold the current size of the list.

List Functions

- **int unLink(int* l, char* name, FILE *p**) - "unlinks" a block from the head of list l. Actually just returns the value of the first position of the array and shifts each value down a location to mimic a list. Prints the operations performed to a file corresponding to FILE pointer p. name is the name of the list, used when printing to the file.

- **void Link(int n, int* l, char* name, FILE *p)** – "links" a block to the tail of list l. links the value n, to the current tail of list l and outputs results to file pointed to by p.

- **void produce_information_in_block(int* b)** - places a random value in the location pointed to by b

- **void use_block_x_to_produce_info_in_y(int* x, int* y)** - uses the value pointed to by x to compute a value that is stored in the location pointed to by y.

- **void consume_information_in_block(int *c)** – sets the value pointed to by c to -1, which is used to denote an "empty" block in this implementation.

- **void printList(int *l, int length, FILE *p)** – prints the list up to a certain length to a file pointed to by p.

- **void initList(int *l, int length, int currSize)** – sets the current size of the lists (0 for list_1 and list_2, N for freelist) and initializes each location up to currSize to -1 (used only for freelist to denote empty blocks).

Marco Duarte
CSCE 5640
October 30, 2017

Project #1

Semaphores

This implementation achieves maximum concurrency and is deadlock-free using seven semaphores: four counting semaphores and three binary semaphores.

- **FlCount -** This semaphore is initialized to N-1, where N is the size of the list, because there can be a scenario where process1 removes all blocks from the freelist without any blocks being on list_2 which causes a deadlock. Counts to N-1 blocks on the freelist.

- **FlCount2 -** Initialized to N, this semaphore is what guarantees maximum concurrency between the processes. Counts the number of blocks on the free list.

- **L1Count** – Initialized to 0, counts how many blocks are on list_1

- **L2Count** – Initialized to 0, counts how many blocks are on list_2

- **useFL** – Initialized to 1, facilitates mutual exclusion on freelist

- **useL1** – Initialized to 1, facilitates mutual exclusion on list_1

- **useL2** – Initialized to 1, facilitates mutual exclusion on list_2

Processes

process1

Process1 must wait if there is only one block on the freelist and process2 has not placed a block on list_2 (FlCount). Once Process2 has placed a block on list_2, process1 waits again if there are no blocks on the freelist (FlCount2). Once it is process1's turn to access the freelist (useFL), it unlinks a block, and wakes up any processes waiting to use the freelist. After producing information, it waits until it can use list_1 (useL1), links to it, and signals processes waiting to use list_1.

process2

Process2 does not have to wait on FlCount since it will not cause a deadlock if it removes the last block of the freelist because it always returns it. Waiting if there are no blocks on the freelist (FlCount2), the process unlinks from it and signals processes waiting to use the freelist. It also signals to process1 that it has taken a block from the freelist so that if there is only one block left it is ok for process1 to take it because the freelist will get blocks from process2 and process3 at some point later on, which avoids a deadlock and also does not overly restrict process1.

process3

This process unlinks from list_2, consumes information, and links the empty block to the freelist.