Deadlock Detection Project

Jordon Biondo

November 7, 2013

1 Design Document

1.1 Data Structures

The main data structure is the process datastructure, it has three threads to simulate a process. A Simulated top level process thread, a main execution thread and a messenger thread. It also holds flags to tell if it is deadlocked, or killed. It also holds a list of owned resources and requested resources. It is shown below /** ,* Simulated process type ,*/ typedef struct { pthread_t simulation; // the simulated process thread id pthread_t worker; // the worker thread, reports deadlock pthread_t messager; // sends/recieves/forwards probes, finds deadlock int messages[1]; // processes communication pipe $\text{pthread}_{\text{mutex}t}$ $\text{msg}_{\text{mutex}}$; // mutex for safe pipe IO int requesting; // index of requested resource bool owning[$\text{RESOURCE}_{\text{LIMIT}}$]; // list of owned resource indices bool active; // is the process active? (used in the config) bool deadlocked; // true when deadlock found bool killed; // when to stop all process threads } $\text{sim}_{\text{process}}$;

1.2 Grammar

Flex and Yacc are used to parse configuration files which allows for easier future extension to the process state configuration files.

1.3 Algorithm

The program follows the same algorithm as described in the specification. After parsing the text, the simulated processes are started and probes begin to be sent. Probes are sent via pipes. Each simulated process has it's own pipe and mutex to control safe IO to that pipe.

[1] DEFINITION NOT FOUND: 2