```c
#ifndef __SEM_H

#include <signal.h>

#define NUM_PROC 64
#define MYPROCS 4

extern int procNum;      // Current Process Index
extern pid_t *pid_table; // Table of Process PIDS

struct sem
{
    char spinlock;             // The Lock
    int max;                   // Max Resources Available
    int free;                  // Actaul Available Resources
    int procInd;               // Index of proc_block for book keeping
    int proc_block[NUM_PROC]; //List of Blocking Processes
    sigset_t mask_block;      //Mask for all signals but SIGUSR1
};

//   Initialize the semaphore *s with the initial count. Initialize
//   any underlying data structures.  sem_init should only be called
//   once in the program (per semaphore).  If called after the
//   semaphore has been used, results are unpredictable.
void sem_init(struct sem *s, int count);

//   Attempt to perform the "P" operation (atomically decrement
//   the semaphore).  If this operation would block, return 0,
//   otherwise return 1.
int sem_try(struct sem *s);

//   Perform the P operation, blocking until successful.
void sem_wait(struct sem *s);

//   Perform the V operation.  If any other tasks were sleeping
//   on this semaphore, wake them by sending a SIGUSR1 to their
//   process id (which is not the same as the virtual processor number).
//   If there are multiple sleepers (this would happen if multiple
//   virtual processors attempt the P operation while the count is <1)
//   then \fBall\fP must be sent the wakeup signal.
void sem_inc(struct sem *s);
#define __SEM_H
#endif
```