```
#include "fifo.h"

int fifo_init(struct fifo *f) {
  f->readptr = 0;
  f->writeptr = 0;
  return sem_init(&f->writeAvailable, MYFIFO_BUFSIZ)
       & sem_init(&f->readAvailable, 0)
       & sem_init(&f->mutex, 1);
}

void fifo_wr(struct fifo *f, unsigned long d) {
  while (1) {
    sem_wait(&f->writeAvailable);
    if (sem_try(&f->mutex)) {
      f->buffer[f->writeptr] = d;
      f->writeptr = (f->writeptr + 1) % MYFIFO_BUFSIZ;
      sem_inc(&f->mutex);
      sem_inc(&f->readAvailable);
      return;
    } else {
      sem_inc(&f->writeAvailable);
    }
  }
}

unsigned long fifo_rd(struct fifo *f) {
  unsigned long val;
  while (1) {
    sem_wait(&f->readAvailable);
    if (sem_try(&f->mutex)) {
      val = f->buffer[f->readptr];
      f->readptr = (f->readptr + 1) % MYFIFO_BUFSIZ;
      sem_inc(&f->mutex);
      sem_inc(&f->writeAvailable);
      return val;
    } else {
      sem_inc(&f->readAvailable);
    }
  }
}
```

```c
#include "sem.h"
#include "tas.h"
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>

unsigned my_procnum;

void empty_handler(int sig) {
  if (sig == SIGINT)
    exit(0);
  return;
}

int sem_init(struct sem *s, int count) {
  s->lock = 0;
  s->count = count;
  memset(s->suspended, 0, sizeof s->suspended);
  return (sigfillset(&s->sig) == 0)
       & (sigdelset(&s->sig, SIGUSR1) == 0)
       & (sigdelset(&s->sig, SIGINT) == 0)
       & (signal(SIGUSR1, empty_handler) != SIG_ERR);
}

int sem_try (struct sem *s) {
  int ret = 0;
  while(tas((char *)&s->lock));
  if(s->count > 0) {
    s->count--;
    ret = 1;
  }
  s->lock = 0;
  return ret;
}
void sem_wait(struct sem *s) {
  while (1) {
    // Wait for lock
    while(tas((char *)&s->lock));
    if (s->count > 0) {
      // Decrement and return
      s->count--;
      s->lock = 0;
      return;
    }

    // Add to suspend list
    s->suspended[my_procnum] = 1;
    s->pids[my_procnum] = getpid();
    s->lock = 0;

    // Wake up when called and try again
    // printf("Sleeping: %u\n", my_procnum);
    sigsuspend(&s->sig);
    s->suspended[my_procnum] = 0;
    // printf("Waking  : %u\n", my_procnum);
  }
}
```

```c
void sem_inc (struct sem *s) {
  while(tas((char *)&s->lock));
  s->count++;
  for (size_t i = 0; i < MAX_PROCESSES; i++) {
    if (s->suspended[i]) {
      kill(s->pids[i], SIGUSR1);
      break;
    }
  }
  s->lock = 0;
}
```

```
#include "sem.h"

#define MYFIFO_BUFSIZ 4096

struct fifo {
  struct sem writeAvailable;
  struct sem readAvailable;
  struct sem mutex;
  unsigned long buffer[MYFIFO_BUFSIZ];
  size_t writeptr;
  size_t readptr;
};

int           fifo_init (struct fifo *f);
void          fifo_wr   (struct fifo *f, unsigned long d);
unsigned long fifo_rd   (struct fifo *f);
```

```
#include <sys/types.h>

#define MAX_PROCESSES 64

// Semaphore structure
struct sem {
  char lock;
  int count;
  sigset_t sig;
  char suspended[MAX_PROCESSES];
  pid_t pids[MAX_PROCESSES];
};

extern unsigned my_procnum;

void empty_handler(int sig);

void wake_waiting(struct sem *s);

int  sem_init(struct sem *s, int count);
int  sem_try (struct sem *s);
void sem_wait(struct sem *s);
void sem_inc (struct sem *s);
```

```
#ifndef _TAS_H_
#define _TAS_H_

int tas(volatile char * lock);

#endif /* _TAS_H_ */
```

```
#include <sys/mman.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>

#include "lib/fifo.h"

#define WRITE_SIZE 100
#define NUM_PROC 8

struct fifo * make_mem() {
  void *m = mmap(0, sizeof(struct fifo), PROT_READ|PROT_WRITE, MAP_ANONYMOUS|MAP
_SHARED, −1, 0);
  if (m == (void *)−1){
    perror("mmap()");
    exit(−1);
  }
  return (struct fifo *) m;
}

int main(void) {
  struct fifo *f = make_mem();
  if (!fifo_init(f)) {
    perror("Could not initialize fifo");
    exit(−1);
  }

  pid_t readpid;
  for(size_t i = 0; i < NUM_PROC; i++) {
    my_procnum = i;
    pid_t pid = fork();
    switch (pid) {
      case −1:
        perror("fork()");
        exit(−1);
      case 0:
        if (my_procnum == 0) {
          unsigned long val;
          for (size_t j = 0; j < WRITE_SIZE*(NUM_PROC−1); j++) {
            val = fifo_rd(f);
            printf("Got %lx\n", val);
          }
          printf("Exiting reader.\n");
        } else {
          unsigned long val = (my_procnum << 16);
          for (unsigned long j = 0; j < WRITE_SIZE; j++) {
            printf("Sent %lx\n", val + j);
            fifo_wr(f, val + j);
          }
          printf("Exiting writer.\n");
        }
        exit(0);
      default:
        if (my_procnum == 0)
          readpid = pid;
        continue;
    }
  }
  waitpid(readpid, NULL, 0);
  exit(0);
}
```

```c
#include <sys/mman.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>

#include "lib/sem.h"

#define NUM_PROC 8u
#define ITER 1000000ul

struct shared {
  struct sem sem;
  unsigned long data;
};

struct shared * make_mem() {
  void *m = mmap(0, sizeof(struct shared), PROT_READ|PROT_WRITE, MAP_ANONYMOUS|M
AP_SHARED, −1, 0);
  if (m == (void *)−1){
    perror("mmap()");
    exit(−1);
  }
  return (struct shared *) m;
}

int main(int argc, char **argv) {
  if (argc != 2) {
    printf("Supply an input option:\n\t1 for normal\n\t2 for mutex\n");
    exit(0);
  }
  int mutex = 0;
  if (argv[1][0] == ’1’)
    mutex = 0;
  else if (argv[1][0] == ’2’)
    mutex = 1;
  else
    printf("Not a valid option.  Assuming no mutex.\n");

  struct shared *s = make_mem();
  if (!sem_init(&s−>sem, 1)) {
    perror("Could not initialize semaphore");
    exit(−1);
  }
  s−>data = 0;

  printf("Spawning %u processes.\n", NUM_PROC);
  for(size_t i = 0; i < NUM_PROC; i++) {
    my_procnum = i;
    pid_t pid = fork();
    switch (pid) {
      case −1:
        perror("fork()");
        exit(−1);
      case 0:
        if (mutex) {
          for (unsigned long count = 0; count < ITER; count++) {
            sem_wait(&s−>sem);
            s->data++;
            sem_inc(&s->sem);
          }
        } else {
          for (unsigned long count = 0; count < 1e6; count++) {
            s->data++;
          }
        }
        printf("Exiting child.\n");
        exit(0);
      default:
        continue;
    }

  }
  for (size_t i = 0; i < NUM_PROC; i++)
    waitpid(s->sem.pids[i], NULL, 0);
  printf("Data has value: %lu\n", s->data);
  printf("Data should be: %lu\n", ITER*NUM_PROC);
  printf("Exiting parent process.\n");
  exit(0);
}
```

```
#include <sys/mman.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>

#include "lib/tas.h"

#define NUM_PROC 8u
#define ITER 1000000ul

struct shared {
  int lock;
  unsigned long data;
};

struct shared * make_mem() {
  void *m = mmap(0, sizeof(struct shared), PROT_READ|PROT_WRITE, MAP_ANONYMOUS|M
AP_SHARED, −1, 0);
  if (m == (void *)−1){
    perror("mmap()");
    exit(−1);
  }
  return (struct shared *) m;
}

int main(int argc, char **argv) {
  if (argc != 2) {
    printf("Supply an input option:\n\t1 for normal\n\t2 for mutex\n");
    exit(0);
  }
  int mutex = 0;
  if (argv[1][0] == ’1’)
    mutex = 0;
  else if (argv[1][0] == ’2’)
    mutex = 1;
  else
    printf("Not a valid option.  Assuming no mutex.\n");

  struct shared *s = make_mem();
  s−>data = 0;
  int child = 0;
  pid_t pids[NUM_PROC];

  printf("Spawning %u processes.\n", NUM_PROC);
  for(size_t i = 0; i < NUM_PROC; i++) {
    pid_t pid = fork();
    switch (pid) {
      case −1:
        perror("fork()");
        exit(−1);
      case 0:
        child = 1;
        goto endloop;
      default:
        pids[i] = pid;
        continue;
    }
  }
  endloop:

  if (child) {
    unsigned long count;

    if (mutex) {
      for (count = 0; count < ITER; count++) {
        while (tas((char *)&s->lock));
        s->data++;
        s->lock = 0;
      }
    } else {
      for (count = 0; count < 1e6; count++) {
        s->data++;
      }
    }
    printf("Exiting child.\n");
    exit(0);
  } else {
    for (size_t i = 0; i < NUM_PROC; i++)
      waitpid(pids[i], NULL, 0);
    printf("Data has value: %lu\n", s->data);
    printf("Data should be: %lu\n", ITER*NUM_PROC);
    printf("Exiting parent process.\n");
    exit(0);
  }
}
```