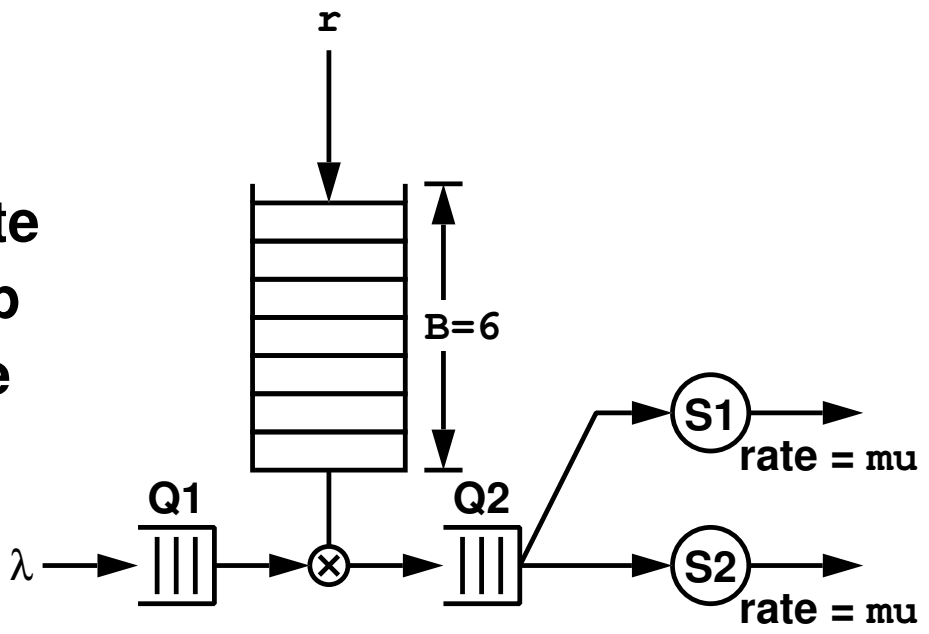
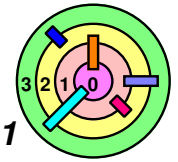
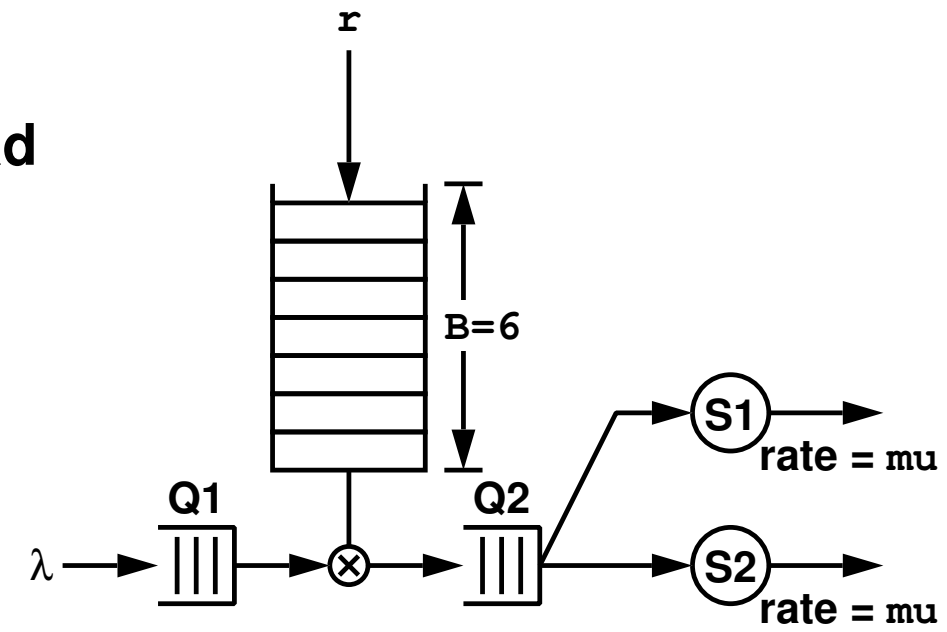
Handling <Cntrl+C> In Warmup #2

- <Cntrl+C>
  - packet arrival thread will stop generating packets and terminate
  - token depositing thread will stop generating tokens and terminate
  - a server thread must finish serving its current packet
  - no more packets or tokens must arrive
  - must print statistics for all packet *completely served by server*
    - need to make sure that packets deleted this way do *not* participate in certain statistics calculation
      - see spec

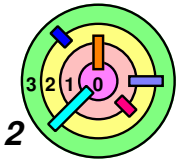




Handling <Cntrl+C> In Warmup #2



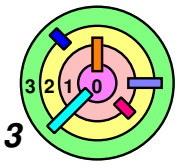
- Many choices to choose from
  - can use the packet arrival thread to catch SIGINT
  - or use the main thread thread to catch SIGINT
  - or use a separate thread just to catch SIGINT
    - this is my preference
    - see `sigwait()` in lecture slides
    - no signal handler!
  - you need to make design decisions and figure out the details
  - you should use the ***pthread cancellation*** mechanism to cancel threads safely
    - understanding cancellation will hopefully prevent you from make some mistakes in kernel assignment #1
      - although kernel cancellation is different



Designate A Thread To Catch A Signal

- Look at the man pages of `pthread_sigmask()` on `nunki` and try to understand the example there
  - designate child thread to handler SIGINT
  - parent thread blocks SIGINT

```
#include <pthread.h>
/* #include <thread.h> */

thread_t user_threadID;
sigset_t new;

void *handler(), interrupt();

main( int argc, char *argv[] )  {
    sigemptyset(&new);
    sigaddset(&new, SIGINT);

    pthread_sigmask(SIG_BLOCK, &new, NULL);
    pthread_create(&user_threadID, NULL, handler, argv[1]);
    pthread_join(user_threadID, NULL);

    printf("thread handler, %d exited\n",user_threadID);
    sleep(2);
    printf("main thread, %d is done\n", thr_self());
} /* end main */
```

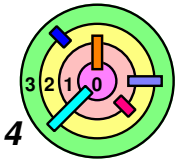


pthread_sigmask()

- Child thread example
  - child thread unblocks SIGINT

```
struct sigaction act;

void *
handler(char argv1[])
{
    act.sa_handler = interrupt;
    sigaction(SIGINT, &act, NULL);
    pthread_sigmask(SIG_UNBLOCK, &new, NULL);
    printf("\n Press CTRL-C to deliver SIGINT\n");
    sleep(8);  /* give user time to hit CTRL-C */
}

void
interrupt(int sig)
{
    printf("thread %d caught signal %d\n", thr_self(), sig);
}
```

  - child thread is designated to handle SIGINT, no other thread will get SIGINT

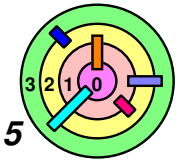


Another Way - No Signal Handler

```
some_state_t state;
sigset_t set;

main() {
  pthread_t thread;
  sigemptyset(&set);
  sigaddset(&set,
            SIGINT);
  sigprocmask(
      SIG_BLOCK,
      &set, 0);
  // main thread
  //      blocks SIGINT
  pthread_create(
      &thread, 0,
      monitor, 0);
  long_running_proc();
}
```

```
void long_running_proc() {
  while (a_long_time) {
    pthread_mutex_lock(&m);
    update_state(&state);
    pthread_mutex_unlock(&m);
    compute_more();
  }
}

void *monitor() {
  int sig;
  while (1) {
    sigwait(&set, &sig);
    pthread_mutex_lock(&m);
    display(&state);
    pthread_mutex_unlock(&m);
  }
  return(0);
}
```

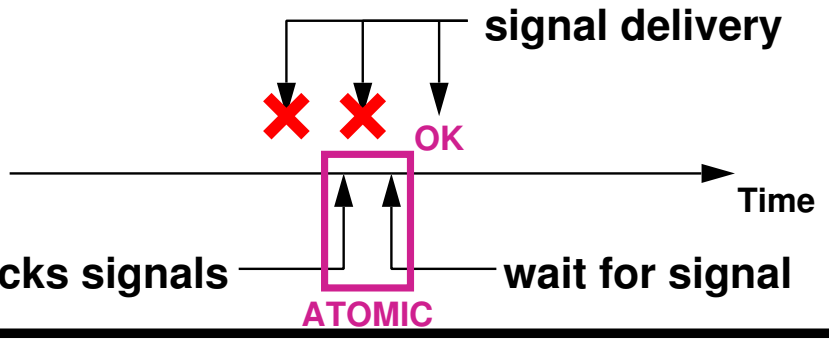
- this is the recommended way to catch SIGINT for warmup2



sigwait()

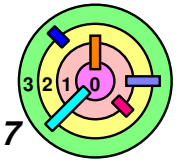
```
int sigwait(sigset_t *set, int *sig)
```

- `sigwait()` blocks until a signal specified in `set` is received
  - return which signal caused it to return in `sig`
  - if you have a signal handler specified for `sig`, it will *not* get invoked when the signal is delivered
    - instead, `sigwait()` will return
- You should make sure that all the threads in your process have these signals blocked!
  - this way, when `sigwait()` is called, the calling thread temporarily becomes the *only* thread in the process who can receive the signal
- `sigwait(set)` *atomically unblocks* signals specified in `set` and *waits* for signal delivery



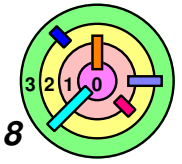
How To Learn New Concepts

- If there is a new concept that you are not familiar with, don't just try to write the final program
  - you won't know where the bugs are because you may not be clear about the concepts at multiple places
- Try writing small programs to test out ideas
  - try one idea at a time
  - use the debugger to get a better understanding of what's going on
  - then compile multiple ideas into one program and see if it works
- Ex:
  - `fork-wait.c`
  - `cat.c`
  - `redirect.c`
  - `thr-term.c`
  - `busywait.c`, `join.c`
  - `deadlock.c`, `trylock.c`
  - `whoopee.c`
  - `status-update.c`
  - `sigblock.c`, `sigwait.c`
  - `direct.c`
  - `cancel.c`



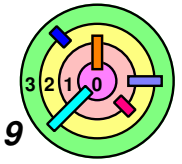
defs.h

```
#ifndef _DEFS_H_
#define _DEFS_H_

#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <signal.h>

#include <pthread.h>

#ifndef NULL
#define NULL 0L
#endif /* ~NULL */

#endif /*_DEFS_H_*/
```



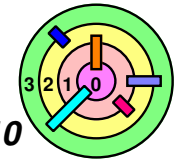
fork-wait.c

```
#include "defs.h"
#define NUM_CHILD 5
int sleep_time[NUM_CHILD], chd_num=0;
int main(int argc, char *argv[])
{
  srand48(time(0));
  for (chd_num=0; chd_num < NUM_CHILD; chd_num++) {
    sleep_time[chd_num] = lrand48() % 5000000;
    if (fork() == 0) {
      int pid=((int)getpid());
      printf("(Child) pid = %1d (0x%08x)\n", pid, pid);
      usleep(sleep_time[chd_num]);
      exit(child_pid+1);
    }
  }
  for (;;) {
    int pid=0, rc=0;
    if ((pid=wait(&rc)) == (-1)) break;
    printf("child %1d exited: 0x%08x.\n", pid, rc);
  }
  return 0;
}
```



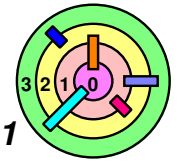
cat.c

```
#include "defs.h"
#define BUFSIZE 1024
int main(int argc, char *argv[])
{
  char buf[BUFSIZE];
  int n=0;
  const char *note="Write failed\n";

  while ((n = read(0, buf, sizeof(buf))) > 0)
    if (write(1, buf, n) != n) {
      (void)write(2, note, strlen(note));
      exit(1);
    }
  return 0;
}
```



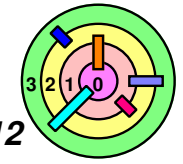
redirect.c

```
#include "defs.h"
int main(int argc, char *argv[])
{
  pid_t pid=(pid_t)0;
  if ((pid=fork()) == 0) {
    close(1);
    if (open("/tmp/Output",
        O_CREAT|O_WRONLY,
        0666) == -1) {
      perror("/tmp/Output");
      exit(1);
    }
    execl("/bin/date", "date", (char*)0);
    exit(1);
  }
  while(pid != wait(0)) ;
  return 0;
}
```



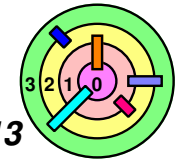
thr-term.c

```
#include "defs.h"
void *child(void *arg)
{
  if (*(int*)arg > 2) pthread_exit((void*)1);
  return((void*)2);
}
int main(int argc, char *argv[])
{
  pthread_t thread;
  void *result=NULL;
  pthread_create(&thread, 0, child, &argc);
  pthread_join(thread, (void**)&result);
  switch ((int)(long)result) {
  case 1: printf("result is 1\n"); break;
  case 2: printf("result is 2\n"); break;
  }
  return 0;
}
```



busywait.c

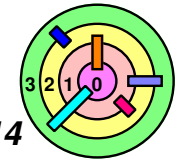
```
#include "defs.h"
#define NUM_THRS 100
int done[NUM_THRS];
pthread_t tid[NUM_THRS];
void *child(void *arg)
{
  int index=(int)(long)(arg);
  usleep(lrand48()%10000000);
  done[index] = 1;
  return 0;
}
int main(int argc, char *argv[])
{
  int i=0;
  memset(done, 0, sizeof(done));
  srand48(0);
  for (i=0; i < NUM_THRS; i++)
    pthread_create(&tid[i], 0, child, (void*)(long)i);
  waitall();
  return 0;
}
```



busywait.c

```
void waitall()
{
  for (;;) {
    int i=0, num_done=0;
    for (i=0; i < NUM_THRS; i++) {
      if (!done[i]) break;
      num_done++;
    }
    if (num_done == NUM_THRS) break;
  }
}
```

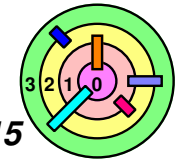
- Why is this busy wait?
  - the main thread is not doing anything useful
- Fix?
  - sleep for 100ms before checking
    - to avoid doing busy wait
    - not really a good solution



join.c

- Real fix
  - join with all threads

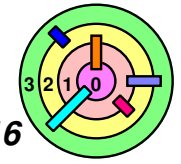
```
void waitall()
{
  int i=0;
  for (i=0; i < NUM_THRS; i++)
    pthread_join(tid[i], 0);
}
```



deadlock.c

Try to deadlock child thread and main thread

```
#include "defs.h"
void *child(void *arg)
{
  for (;;) { proc1(); }
  return((void*)0);
}

int main(int argc, char *argv[])
{
  pthread_t thread;
  srand48(time(0));
  pthread_create(&thread, 0, child, 0);
  for (;;) { proc2(); }
  return 0;
}
```

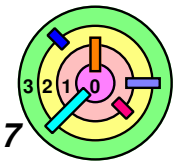


deadlock.c

```
pthread_mutex_t m1=PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t m2=PTHREAD_MUTEX_INITIALIZER;

void proc1() {
  pthread_mutex_lock(&m1);
  pthread_mutex_lock(&m2);
  printf("1");
  fflush(stdout);
  pthread_mutex_unlock(&m2);
  pthread_mutex_unlock(&m1);
  usleep(100000);
}
```

```
void proc2() {
  pthread_mutex_lock(&m2);
  pthread_mutex_lock(&m1);
  printf("2");
  fflush(stdout);
  pthread_mutex_unlock(&m1);
  pthread_mutex_unlock(&m2);
  usleep(100000);
}
```

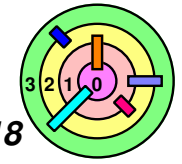
- Why no deadlock?
  - threads are alternating
- How to make it deadlock?
  - call `printf("-")` after locking `m1` in `proc1()` and call `printf("+")` after locking `m2` in `proc2()`
    - deadlock right away! (why?)



trylock.c

How to use `trylock()` to avoid deadlock

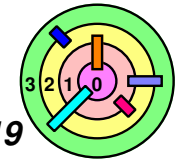
```
void proc2() {
  while (1) {
    pthread_mutex_lock(&m2);
    printf("+");
    fflush(stdout);
    if (!pthread_mutex_trylock(&m1))
      break;
    pthread_mutex_unlock(&m2);
  }
  printf("2");
  fflush(stdout);
  pthread_mutex_unlock(&m1);
  pthread_mutex_unlock(&m2);
  usleep(100000);
}
```



whoopee.c

Let's catch SIGINT

```
#include "defs.h"
void handler(int signo)
{
  printf("Got signal %1d.  Whoopee!!\n", signo);
}
int main(int argc, char *argv[])
{
  sigset(SIGINT, handler);
  for (;;) { }
  return 1;
}
```



status-update.c

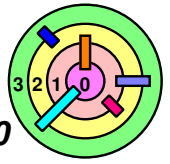
```
#include "defs.h"
typedef struct foo {
  int x, y;
} my_state;
my_state state;
int main() {
  state.x = state.y = 0;
  sigset(SIGINT, handler);
  long_running_proc();
  return 0;
}
```

```
void long_running_proc() {
  int i=0;
  for (i=0; i < 100; i++) {
    update_state(&state);
    compute_more();
  }
}
void handler(int signo) {
  display(&state);
}
```

```
void update_state(
    my_state *ptr) {
  ptr->x++;
  usleep(100000);
  ptr->y++;
}
```

```
void display(my_state *ptr) {
  printf("x = %1d\n", ptr->x);
  usleep(1000);
  printf("y = %1d\n", ptr->y);
}
```

```
void compute_more() { usleep(100000); }
```

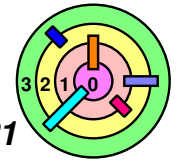


sigblock.c

- Let's block SIGINT

```
#include "defs.h"
typedef struct foo {
  int x, y;
} my_state;
my_state state;
sigset_t set;
int main() {
  state.x = state.y = 0;
  sigemptyset(&set);
  sigaddset(&set, SIGINT);
  sigset(SIGINT, handler);
  long_running_proc();
  return 0;
}
```

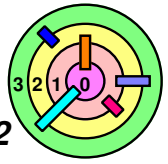
```
void long_running_proc() {
  int i=0;
  for (i=0; i < 100; i++) {
    sigset_t old_set;
    sigprocmask(SIG_BLOCK,
      &set, &old_set);
    update_state(&state);
    sigprocmask(SIG_SETMASK,
      &old_set, 0);
    compute_more();
  }
}
void handler(int signo) {
  display(&state);
}
```



sigwait.c

```
#include "defs.h"
typedef struct foo {
  int x, y;
} my_state;
my_state state;
sigset_t set;
pthread_mutex_t m=
PTHREAD_MUTEX_INITIALIZER;
int main() {
  pthread_t thr;
  state.x = state.y = 0;
  sigemptyset(&set);
  sigaddset(&set, SIGINT);
  sigprocmask(SIG_BLOCK,
    &set, 0);
  pthread_create(&thr, 0,
    monitor, 0);
  long_running_proc();
  return 0;
}
```

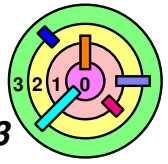
```
void long_running_proc() {
  int i=0;
  for (i=0; i < 100; i++) {
    pthread_mutex_lock(&m);
    update_state(&state);
    pthread_mutex_unlock(&m);
    compute_more();
  }
}
void *monitor(viod *arg) {
  int sig=0;
  for (;;) {
    sigwait(&set, &sig);
    pthread_mutex_lock(&m);
    display(&state);
    pthread_mutex_unlock(&m);
  }
  return 0;
}
```



direct.c

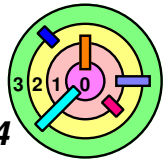
- Direct a thread to catch SIGINT
  - see "man pthread_sigmask" on nunki

```
#include "defs.h"
pthread_t thr;
sigset_t set;

void handler(int sig)
{
  printf("\nthread 0x%08x caught signal %d\n",
    (int)pthread_self(), sig);
}

void *child(void *arg)
{
  sigset(SIGINT, handler);
  pthread_sigmask(SIG_UNBLOCK, &set, 0);
  printf("Press <Cntrl+C> to deliver SIGINT: ");
  fflush(stdout);
  sleep(8);
  return 0;
}
```



direct.c

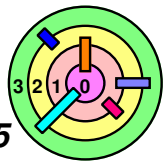
```
int main(int argc, char *argv[])
{
  sigemptyset(&set);
  sigaddset(&set, SIGINT);
  pthread_sigmask(SIG_BLOCK, &set, 0);
  pthread_create(&thr, 0, child, 0);
  pthread_join(thr, 0);
  printf("thread 0x%08x exited\n", (int)thr);
  printf("thread 0x%08x (main) is done\n",
    (int)pthread_self());
  return 0;
}
```



cancel.c

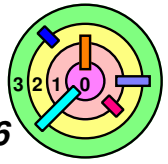
Cancellation

```
#include "defs.h"
#define NUM_THRS 10
pthread_t tid[NUM_THRS];
sigset_t set;
void cleanup(void *arg)
{
  int index=(int)(long)(arg);
  printf("Clean up thread %1d\n", index);
}
void *child(void *arg)
{
  pthread_cleanup_push(cleanup, arg);
  for (;;) {
    usleep(lrand48()%1000000);
  }
  pthread_cleanup_pop(0);
  return 0;
}
```



cancel.c

```
void waitall()
{
  int i=0;
  for (i=0; i < NUM_THRS; i++)
    pthread_join(tid[i], 0);
}
void *monitor(void *arg) {
  int i=0, sig=0;
  printf("Press <Cntrl+C>: ");
  fflush(stdout);
  sigwait(&set, &sig);
  printf("\nGot signal %1d\n", sig);
  for (i=0; i < NUM_THRS; i++) {
    while (tid[i] == 0) {
      usleep(100000);
    }
    pthread_cancel(tid[i]);
  }
  return 0;
}
```



cancel.c

```
int main(int argc, char *argv[])
{
  pthread_t thr;
  int i=0;
  srand48(time(0));
  sigemptyset(&set);
  sigaddset(&set, SIGINT);
  sigprocmask(SIG_BLOCK, &set, 0);
  for (i=0; i < NUM_THRS; i++) {
    pthread_create(&tid[i], 0, child, (void*)(long)i);
  }
  pthread_create(&thr, 0, monitor, 0);
  waitall();
  pthread_join(thr, 0);
  return 0;
}
```