ELEC 377
Operating Systems

Week 7 Lab

Lab 3

- Purpose 1 – implement synchronization
- Purpose 2 – shared memory on Unix
- Purpose 3 – separate compilation

Lab 3

- Skeleton Code is in git and on the handouts page of the web site
  ◊ 6 files:
  -producer.c, consumer.c
  -common.h, common.c
  -makefile, meminit.c

- Makefile is used to simplify building systems
  make - with no args builds entire system
  make xxx - builds xxx

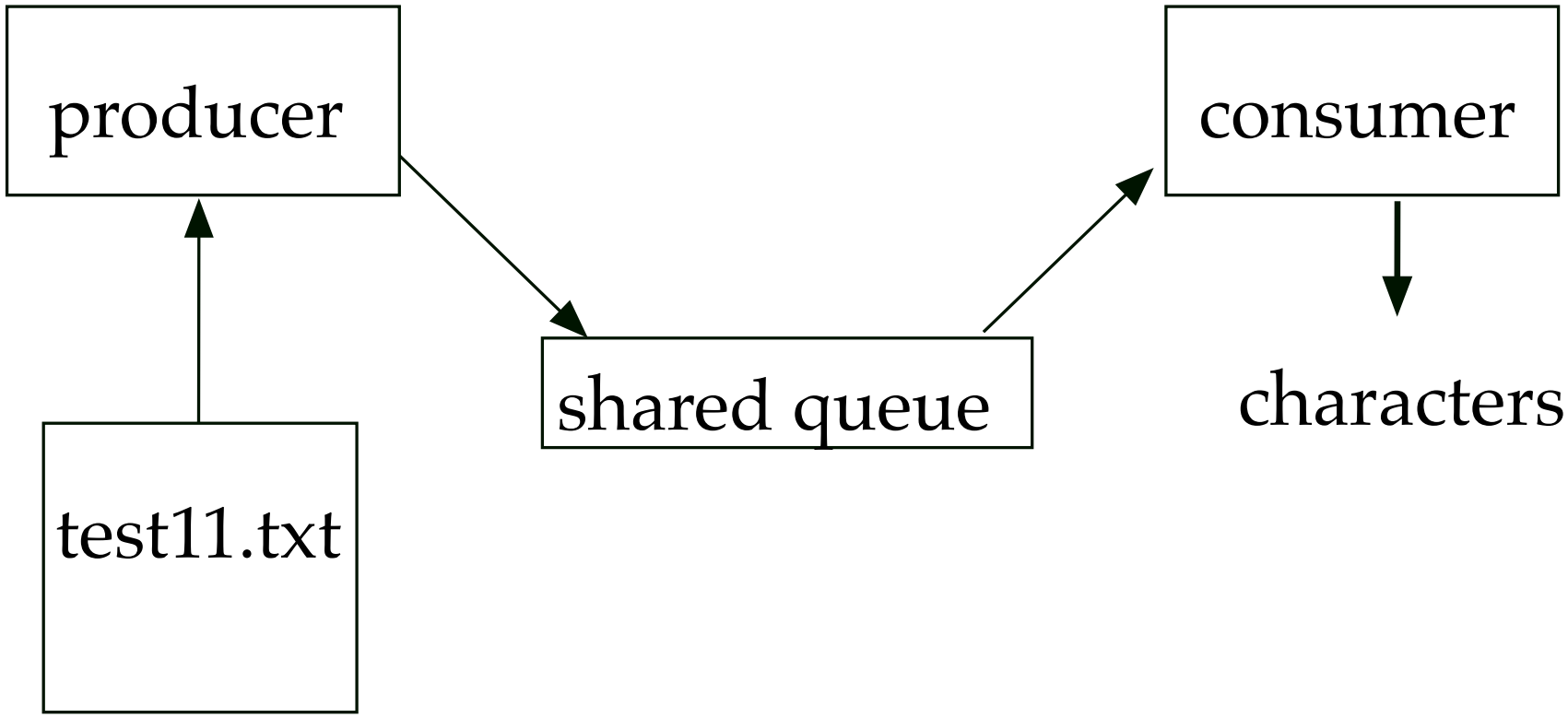
Lab 3 - User Level Code

- Programs run at the user level
- Not Kernel dependent
- Can be done on any version of Unix with shared memory segments
  - Linux
  - Sun
  - Mac Os X

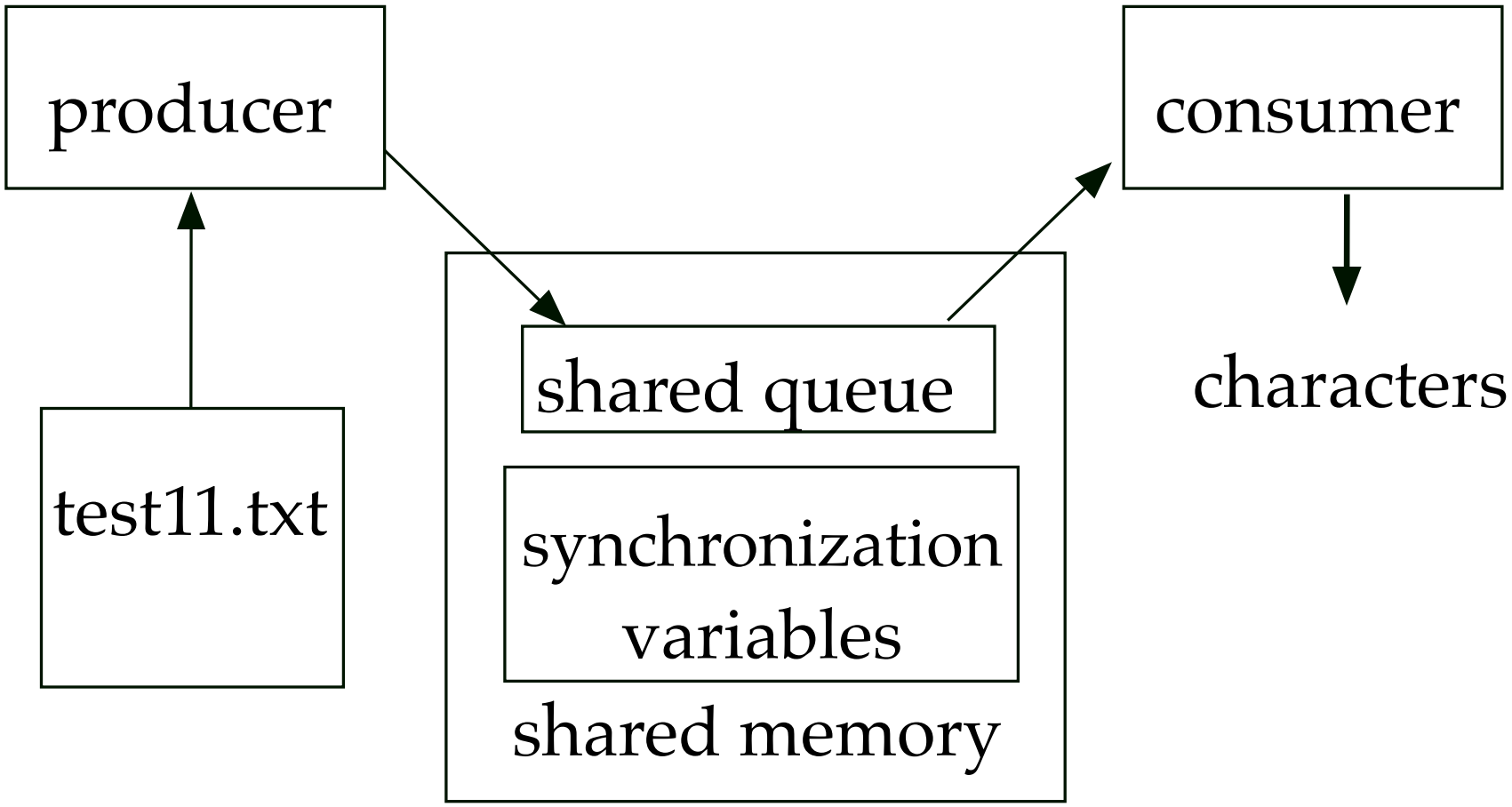
Lab 3 - System Structure

- Producer, Consumer
- Producer reads a file of characters and passes it to the consumer through the shared memory
- Consumer reads characters through the shared memory and prints them out

Lab 3 - System Structure



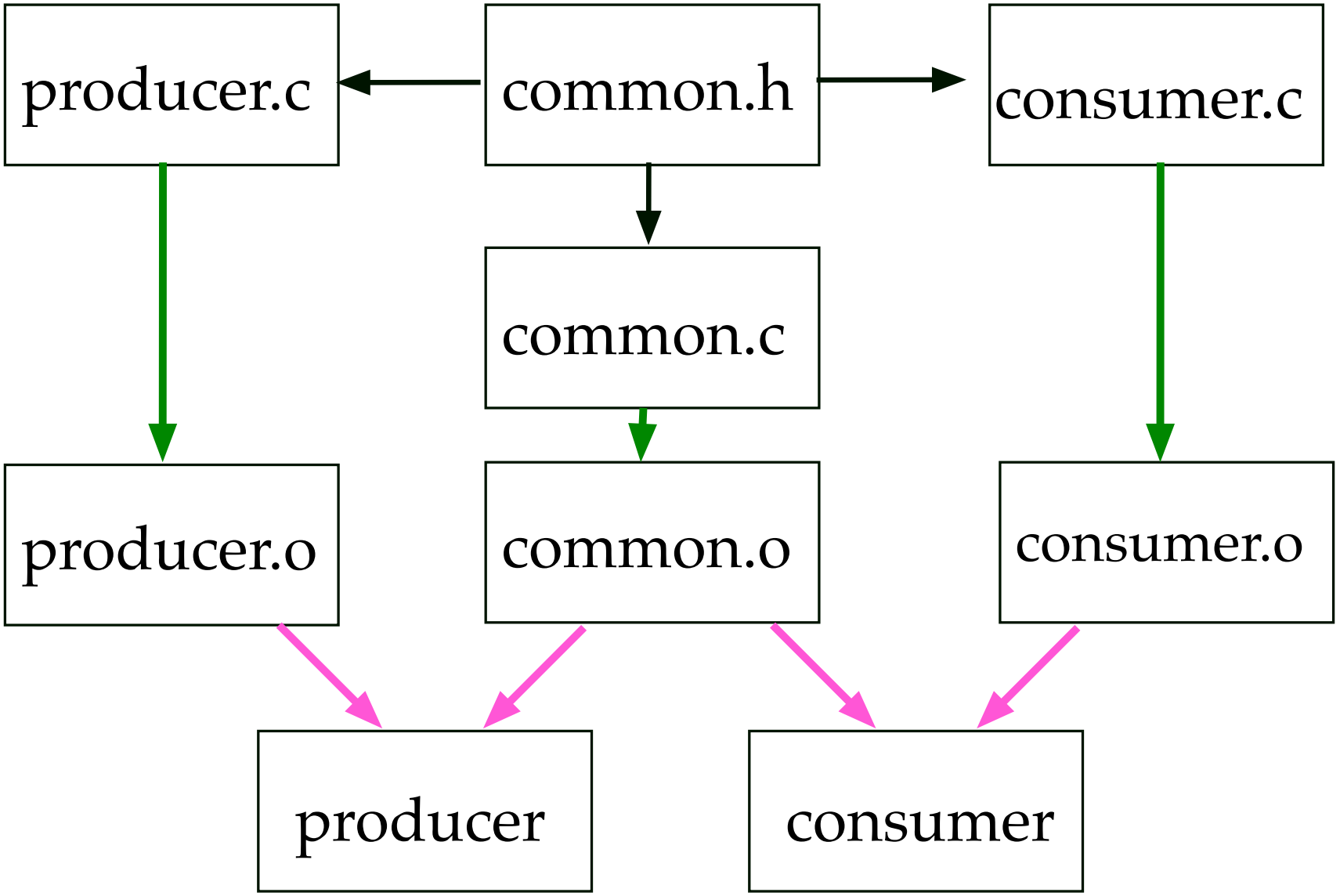
Lab 3 - System Structure



Lab 3 - Compile Time Structure

- mutexes are common code
  - ◊ same code for both producer and consumer
  - ◊ put in a common file so that both can use same routines
- also need to impose structure on the share memory
  - ◊ shmat returns void *
    - pointer to arbitrary memory
  - ◊ both consumer and producer have to have the same view of the shared memory
    - structure definition is common to producer and consumer

Lab 3 - Compile Structure



Lab 3 - Shared Memory Structure

#define NUMPROCS 5 // or an appropriate number

```
struct shared {
   /* synchronization variables */
   int waiting[NUMPROCS];
   int lock;

   int numProducers;
   /* queue variables */
   char buffer[BUFFSIZE];
   int in;
   int out;
   int count;
}
```

Lab 3 - Common.c

```
void getMutex(int pid){
}

void releaseMutex(int pid){
}
```

use the shared pointer to get at the shared synchronization control variables!!

pid is the same as *i* in the algorithm

Lab 3 - When to stop

- Use a flag in the shared memory structure to indicate the number of producers

- Producers increment and decrement the number of products (guarded by a mutex).

- Consumer
  - ◊ When the queue is empty, check if the number of producers is zero.

Lab 3 - What you have to do

- common.c
  - ◊ write getMutex and releaseMutex based on the pass the key algorithm solution
- producer.c
  - ◊ read data from file
  - ◊ add data to queue
- consumer.c
  - ◊ read data from queue
  - ◊ write to std out
- Transfer data one item at a time
- Nested Loops in both producer and consumer
- ***All*** access to shared data is guarded by the semaphores
- No I/O in the critical section!!

Lab 3 - Shared Memory Structure

using more than one producer, or consumer – variant of reader/writer problem

```
int main(int argc, char argv[])

./producer 1

argc = 2
argv[0] = "./producer"
argv[1] = "1"

int pid = atoi(argv[1]);
```

Lab 3 - producer

```
get pid from command line
getMutex(pid)
   numProducers++
releaseMutex(pid)
loop (while not eof)
   get character
   stored = false
   loop while stored = false
      getMutex(pid)
         if room add character, stored = true
      releaseMutex(pid)
   end loop
end loop
getMutex(pid)
NumProducers--, if numProducers is 0 end of file (guarded)
releaseMutex(pid)
```

Lab 3 - When to stop

- Consumer
  - ◊ When the queue is empty, check the number of producers. If 0, then no more data.

Lab 3 -Testing

- Think about your data
  ◊ your data should prove that
  - all of the data is transferred
  - only the data is transferred
  - the order of the data is preserved
  - no duplicates are introduced.