A Dobby Walk Through

- Learning Objectives
  - Solve Dobby-the-elf problem using locks and CVs
  - Develop comfort using mutexes and condition variables
  - Develop intuition for when condition variables are appropriate
  - Understand the relationship between a condition, a condition variable and a mutex.

Condition Variable Review

- Condition variable:
  - A construct that lets you atomically release a mutex and wait.
- Works in cooperation with:
  - A mutex that protects shared state.
  - A condition that relies on some of the shared state protected by the mutex.

But wait!
How did we know we needed a CV?

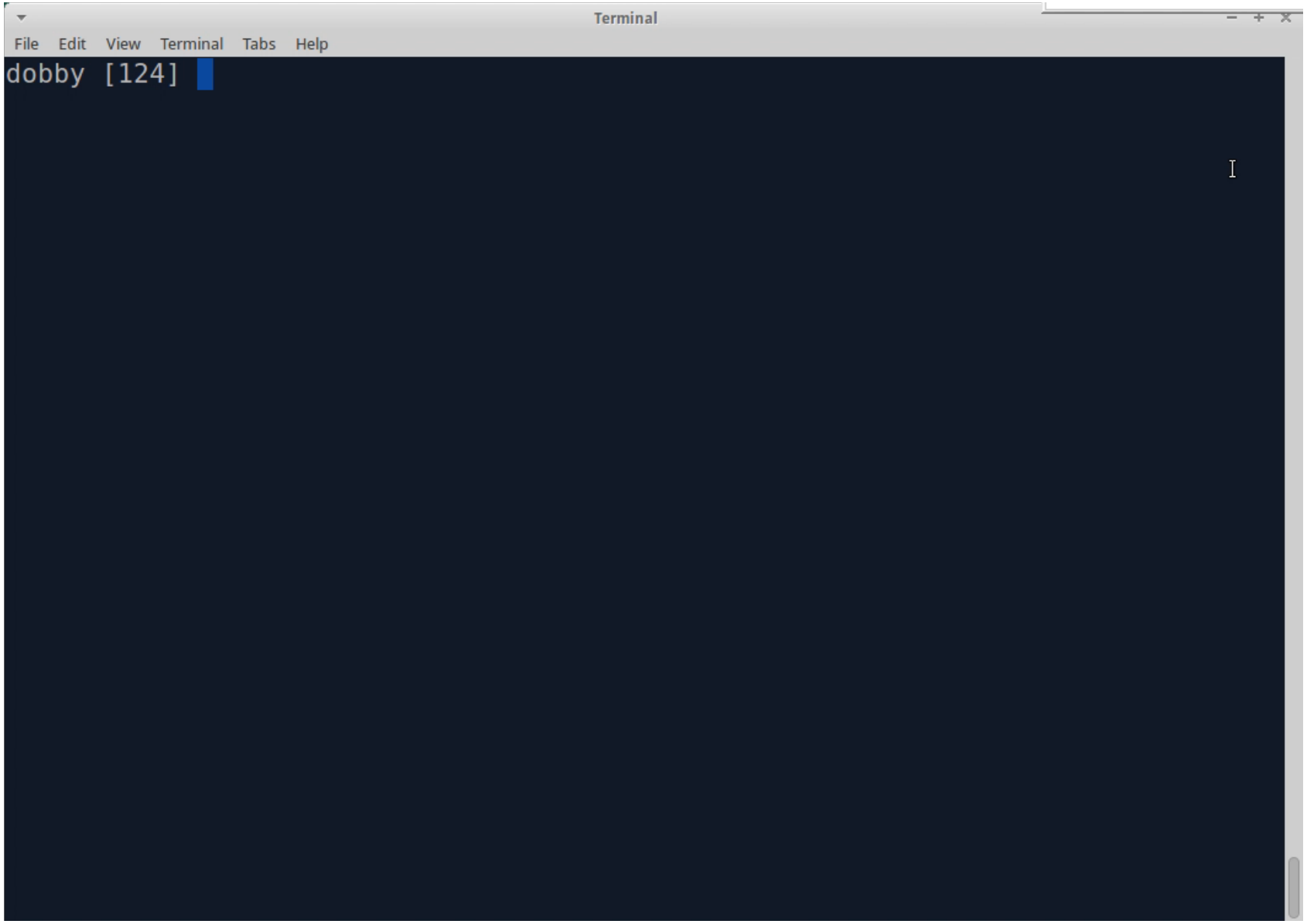
1. The problem statement told you to use locks and CVs: “Since this is from last week, limit yourselves to mutexes and condition variables for this part.”
2. Aside from that: what are you trying to accomplish?
   - “When an elf has completed all of its tasks, the supervisor dismisses the elf by saying "Thanks for your work, Elf N!" where N corresponds to the elf's ID number.”
   - Put another way, Dobby is waiting for elves to complete work. So, what mechanism do we have to make Dobby wait?
   - A CV!

Where there's a CV …

- There is a condition
  - We want to wait until an elf exits, so we're going to have to have something like:

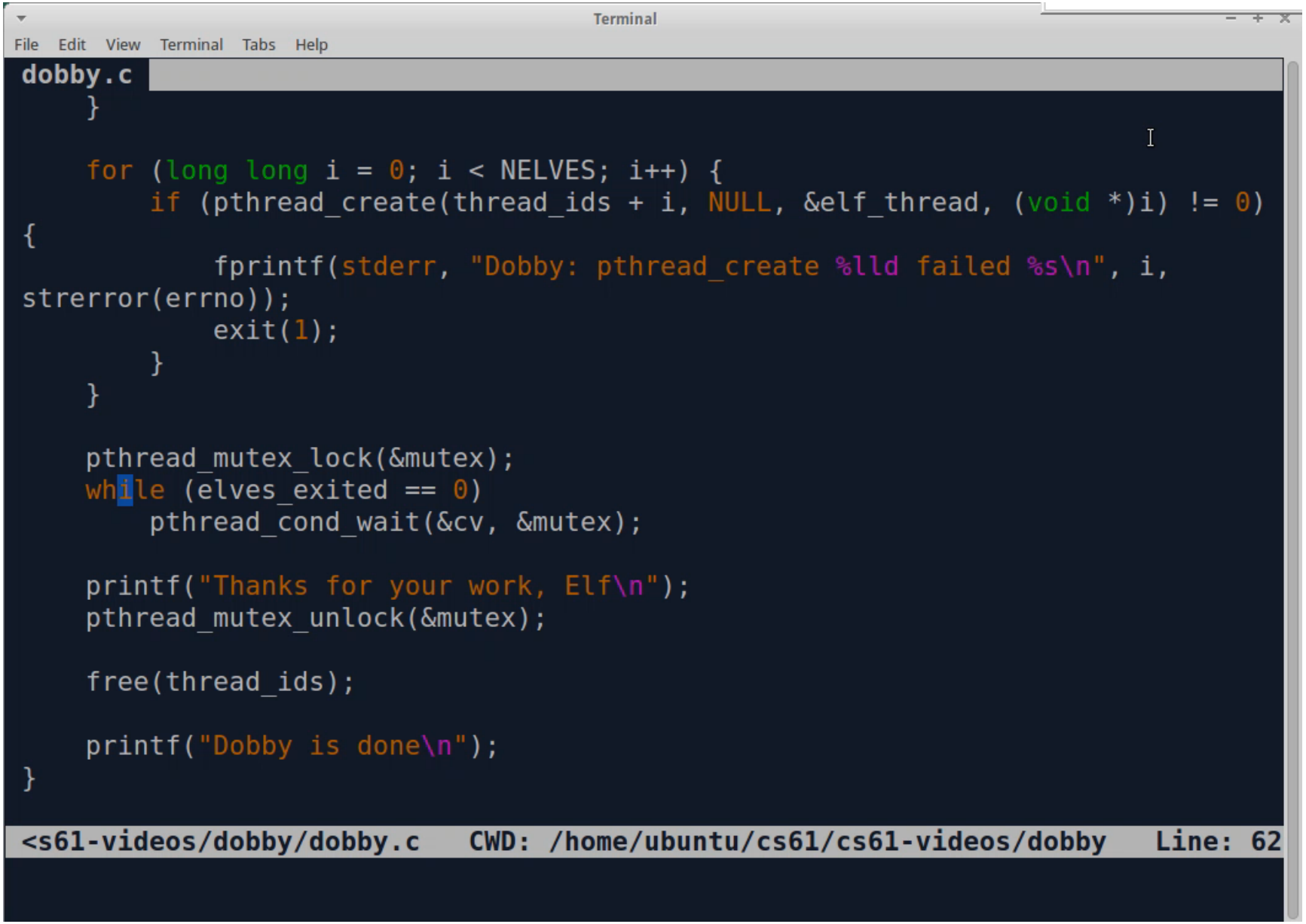
```
pthread_mutex_lock(&mutex);
while (no elf has exited)
    pthread_cond_wait(&cv, &mutex);
```

- There is a mutex
  - The mutex needs to protect whatever we maintain to determine if an elf has exited.


Terminal
File Edit View Terminal Tabs Help
dobby [124]

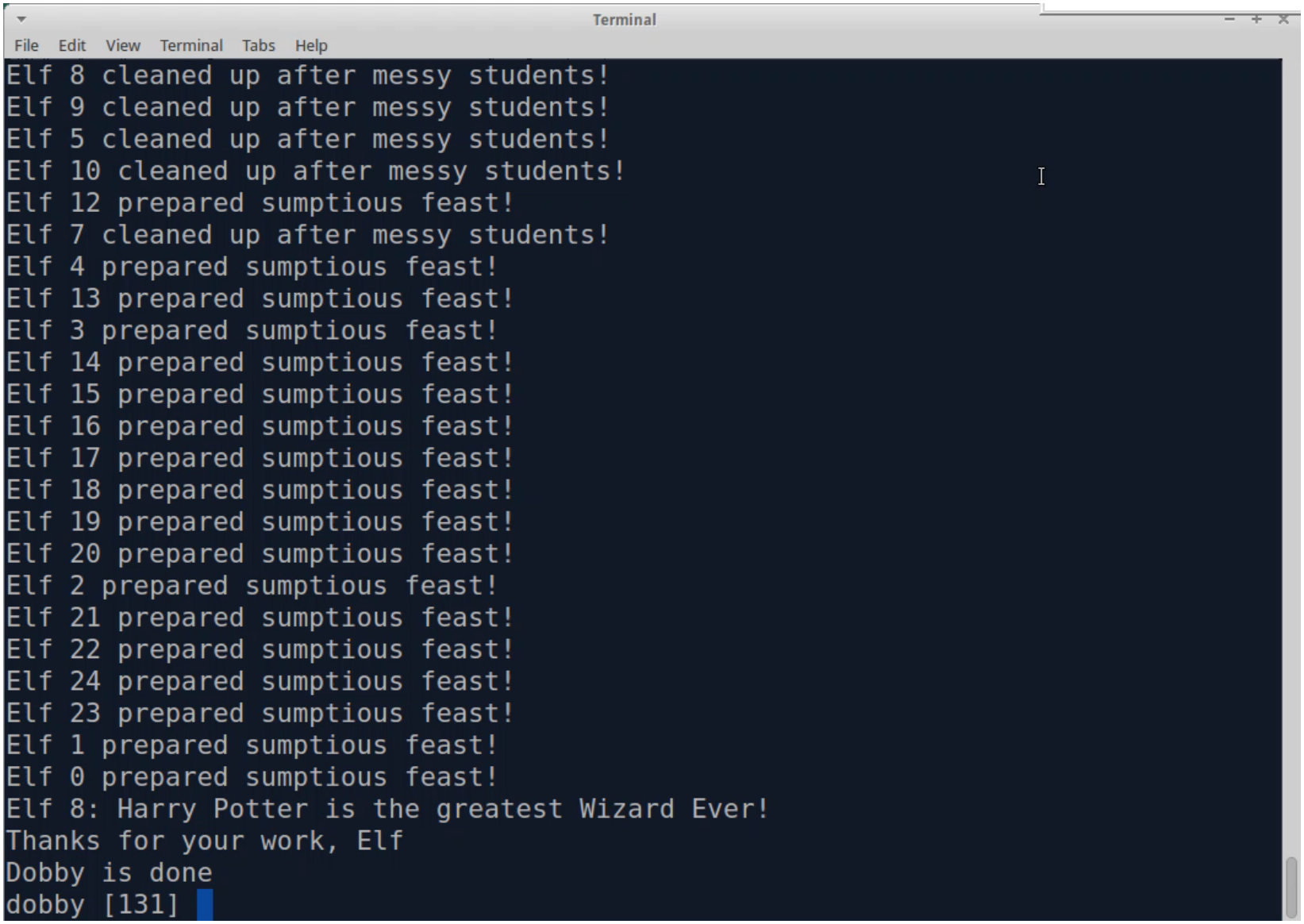
What's next?

- Dobby can now wait, but how do we communicate to Dobby that an elf has exited?
  - We have to make the condition on which Dobby is waiting change
  - We have to signal Dobby's CV.
- Let's work on our elf code!


Terminal
File Edit View Terminal Tabs Help
dobby.c
}
for (long long i = 0; i < NELVES; i++) {
if (pthread_create(thread_ids + i, NULL, &elf_thread, (void *)i) != 0)
{
fprintf(stderr, "Dobby: pthread_create %lld failed %s\n", i,
strerror(errno));
exit(1);
}
}
pthread_mutex_lock(&mutex);
while (elves_exited == 0)
pthread_cond_wait(&cv, &mutex);
printf("Thanks for your work, Elf\n");
pthread_mutex_unlock(&mutex);
free(thread_ids);
printf("Dobby is done\n");
}
<s61-videos/dobby/dobby.c CWD: /home/ubuntu/cs61/cs61-videos/dobby Line: 62

Which Elf Exited?

- Dobby needs to know which elf to dismiss.
- How can the elves communicate this information to Dobby?
  - One variable doesn't work as there may be multiple elves exiting
  - We need a data structure that can hold multiple elf IDs
    - Linked list
    - Array


Terminal
File Edit View Terminal Tabs Help
Elf 8 cleaned up after messy students!
Elf 9 cleaned up after messy students!
Elf 5 cleaned up after messy students!
Elf 10 cleaned up after messy students!
Elf 12 prepared sumptious feast!
Elf 7 cleaned up after messy students!
Elf 4 prepared sumptious feast!
Elf 13 prepared sumptious feast!
Elf 3 prepared sumptious feast!
Elf 14 prepared sumptious feast!
Elf 15 prepared sumptious feast!
Elf 16 prepared sumptious feast!
Elf 17 prepared sumptious feast!
Elf 18 prepared sumptious feast!
Elf 19 prepared sumptious feast!
Elf 20 prepared sumptious feast!
Elf 2 prepared sumptious feast!
Elf 21 prepared sumptious feast!
Elf 22 prepared sumptious feast!
Elf 24 prepared sumptious feast!
Elf 23 prepared sumptious feast!
Elf 1 prepared sumptious feast!
Elf 0 prepared sumptious feast!
Elf 8: Harry Potter is the greatest Wizard Ever!
Thanks for your work, Elf
Dobby is done
dobby [131]

Wrapping Up

- How did we solve the problem:
  1. Identify the mechanism (Dobby has to wait for an elf to exit; that sounds like a CV!)
  2. Identify the shared state (We need something to tell us if an elf has exited)
  3. We need Dobby to loop on the condition, blocking using the CV
  4. We need the elf thread to update the shared state to reflect that it has exited.
  5. Add array to keep track of elf statuses.