Linked Lists

Kernel Linked Lists

- The Linux kernel contains a variety of data structures
  - Lists, hash tables, trees, etc
- Kernel linked lists are doubly linked and circular
  - Allows easy and quick traversal in both directions

Use Kernel Lists?

- Pros
  - Safer/quicker than own ad-hoc implementation
  - Comes with several ready functions (list.h)
  - Relatively easy to set up a FIFO queue
- Cons
  - Using them can be tricky
  - If you use the wrong function you could crash the kernel
    - e.g. deleting items while traversing list

Declaring a list

```
struct list_head {
  struct list_head *next;
  struct list_head *prev;
};

/* Declare the start of the list */
/* can optionally be within a struct */
struct list_head example_list

/* Initialize the list to empty */
INIT_LIST_HEAD(&example_list);
```

Embedding a list_head

```
/* Object has to have a list_head embedded in it */
/* Compared to making a list of object pointers directly */
typedef struct item {
  struct list_head list
  int num;
} Item;


/* Add an item to end of our example_list*/
Item *item;
item = kmalloc(sizeof(Item),  __GFP_RECLAIM);
item→num = 0;
list_add_tail(&item→list, &example_list);
```

Traversing a list
Fast

```
/* Declare some temporary pointers */
struct list_head *temp;
struct list_head *dummy; //not used here
Item *item;

/* Use this for read-only access */
list_for_each(temp, &example_list) {
  /* Use this to get the surrounding struct */
  item = list_entry(temp, Item, list);

  //can access item->num
}
```

Traversing a list Safe

```
/* Declare some temporary pointers */
struct list_head *temp;
struct list_head *dummy;
Item *item;

/* Use this if you need to change pointers */
list_for_each_safe(temp, dummy, &example_list) {
  /* Use this to get the surrounding struct */
  item = list_entry(temp, Item, list);

  //can access item->num
}
```

Traversing a list
Without list_entry call

```
/* Declare some temporary pointers */
struct list_head *temp; //not used here
struct list_head *dummy; //not used here
Item *item;

/* Use this to get entry immediately (not “safe”) */
list_for_each_entry(item, &example_list, list) {
  //can access item->num



}
```

Moving List Items

```
/* Declare some temporary pointers */
struct list_head *temp;
struct list_head *dummy;
Item *item;

/* Use this since you need to change the pointers */
list_for_each_safe(temp, dummy, &example_list) {
  /* no need to get the entry */
  /* unless you are checking against a condition */

  list_move_tail(temp, &another_example_list);
}
```

Removing List Items

```
/* Declare some temporary pointers */
struct list_head *temp;
struct list_head *dummy;
Item *item;

/* Use this since you need to change the pointers */
list_for_each_safe(temp, dummy, &example_list) {
  item = list_entry(temp, Item, list);

  list_del(temp); /* init ver also reinits list */
  kfree(item); /* remember to free alloced data */
}
```

Other Kernel List Functions?

- Look in include/list.h
  - Contains forward/backward, front/back, safe/unsafe, with/without entry, etc versions
- Look at provided example4 on Canvas
  - Small proc module
  - Randomly creates new animals and adds to list on proc open
  - Displays list stats on proc read
  - Does move and remove on rmmod and prints stats to syslog
  - Also shows how to print many things to a proc file