Fragmentation
As we develop various systems for storage, we want to minimize fragmentation.
- [Fragmentation]:
- [Internal Fragmentation]:
- [External Fragmentation]:

Fragmentation Example in Heap Memory:

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Unallocated (3072 bytes)</td>
<td>Used 1024 bytes</td>
<td>Used 1024 bytes</td>
<td>Used 2048 bytes</td>
</tr>
<tr>
<td></td>
<td>Free 1024 bytes</td>
<td>Free 2048 bytes</td>
<td>Used Data 2048 bytes</td>
<td></td>
</tr>
</tbody>
</table>

Start of Heap

Computer Peripherals
- Every other piece of hardware we consider to be a “peripheral”.
- Interface managed by the ___________________.
  - ...and managed using ___________________.
- Examples:

Threads: The Unit of Computation in an Operating System
As a programmer, the single most important construct in an Operating System is a thread.

- Every thread has a **program counter**, a pointer that stores the next instruction to be read by a program.
- A ________________ is an organization of one or more threads in the same context. A simple process has only one thread.
- In C, the initial thread is called the ___________________.
  - It is what starts running your main() function!

Creating Additional Threads in C
The pthread library is the POSIX thread library allowing you to create additional threads beyond the main thread.

Creating a new thread is a complex call with four arguments:

```c
int pthread_create(
    pthread_t *thread,   /* thread struct */
    const pthread_attr_t *attr, /* usually NULL */
    void *(*start_routine) (void *), /* start func */
    void *arg /* thread start arg */
);
```

The start_routine has a very interesting type signature:

```c
void **(start_routine) (void *)
```

This signature is a **function pointer** (“functor”) and is the syntax we can use to pass a pointer to a function. Therefore, the third argument into pthread_create must be a function with the following prototype:

```c
void *________(void *ptr);
```

...you can use any name for the function name.
### Example: Launching Fifteen Threads

**07/fifteen-threads.c**

```c
#include <stdio.h>
#include <pthread.h>
#include <stdlib.h>

const int num_threads = 15;

void *thread_start(void *ptr) {
    int id = *((int *)ptr);
    printf("Thread %d running...
", id);
    return NULL;
}

int main(int argc, char *argv[]) {
    // Create threads:
    int i;
    pthread_t tid[num_threads];
    for (i = 0; i < num_threads; i++) {
        int *val = malloc(sizeof(int));
        *val = i;
        pthread_create(&tid[i], NULL, 
                       thread_start, (void *)val);
    }
    printf("Done!
");
    return 0;
}
```

**Q1:** What is the expected output of this program?

**Q2:** What actually happens?

**Q3:** What do we know about threads in C?

### Example: Joining Threads

**07/fifteen-join.c**

```c
int main(int argc, char *argv[]) {
    // Create threads:
    int i;
    pthread_t tid[num_threads];
    for (i = 0; i < num_threads; i++) {
        int *val = malloc(sizeof(int));
        *val = i;
        pthread_create(&tid[i], NULL, 
                       thread_start, (void *)val);
    }
    // Joining Threads
    for (i = 0; i < num_threads; i++) {
        pthread_join(tid[i], NULL);
    }
    printf("Done!
");
    return 0;
}
```

In the above program, we use `pthread_join`. This call will block the CPU from running the program further until the specified thread has finished and returned.

**Q4:** What happens in this program?

**Q5:** Does the order vary each time we run it? What is happening?

**Q6:** What can we say about the relationship between “Done” and “Thread %d running...” lines?