CSCE 613: Project 2 Design

Student: Caio Duarte Diniz Monteiro

Date: 02/05/16

FramePool class variables

The frame pool is responsible for managing the physical memory frame allocation for both processes and the kernel. While we have different frame pool instances for the kernel and for the process, the *release_frame* method is static, thus, the *FramePool* class needs to keep track of all the existent pools.

The *FramePool* class have 4 static variables, two of them to hold the addresses of the kernel and process frames bitmaps and other two that holds the info frame numbers for the kernel and process pools.

```
/* kernel frame bitmap and frame number that stores management info*/
static unsigned char* kernel_frame_bitmap;
static unsigned long kernel_info_frame_no;

/* process frame bitmap and frame number that stores management info*/
static unsigned char* proc_frame_bitmap;
static unsigned long proc_info_frame_no;
```

Furthermore, the *FramePool* class has more 4 variables that holds all the information needed by a particular instance of this class.

```
unsigned char* frame_bitmap;
unsigned long info_frame_no;
unsigned long n_frames;
unsigned long base_frame_no;
```

The *frame_bitmap* holds the address of the bitmap for the current instance, *info_frame_no* holds the frame number where the bitmap is stored, *n_frames* tells how many frames this pool is responsible to manage, and the *base_frame_no* tells the starting frame of the current frame pool.

FramePool initialization

```
FramePool(unsigned long _base_frame_no,
    unsigned long _nframes,
    unsigned long _info_frame_no);
```

The *FramePool* class constructor is responsible for initializing the frame status bitmap and store all the pertinent info about the pool (e.g. base frame for the pool, pool size, and number of frames). Since this class is used for both Kernel and Process pools, the first step is to identify which kind of pool it is being created.

If it is a Kernel Pool the following general steps apply:

- Store the frame number where the bitmap is kept. This comes from the third argument of the function, if it is a zero then the frame number is going to be the same as the base frame number of the Kernel.
- Allocate the frame bitmap to the appropriate address (Kernel base address if the third parameter is 0 or the address of this frame if it is different than 0).
- Populate the frame status bitmap, all entries but the one where the bitmap is stored is going to be 0.
- Set the instance variable values.

Otherwise, if it is a Process pool the following steps apply:

- Store the frame number where the bitmap is kept. Similarly to the Kernel pool, this value also comes from the third argument of the function, the difference is that if the value is 0, then the first available frame on the Kernel pool is going to be used.
- Allocate the frame bitmap to the appropriate address, considering the address of the frame used in the previous step.
- Populate the frame status bitmap. Since the bitmap is kept on the Kernel process pool, then we can initialize all the values of the bitmap to 0.
- Set the instance variable values.

FramePool class functions

In order to manage the physical memory allocation, the pool must be able to perform three actions: (i) allocate a new frame; (ii) release an allocated frame; (iii) protect specific parts of the memory.

get_frame()

```
unsigned long get_frame();
```

The *get_frame* function is responsible to traverse the frame status bitmap looking for an available frame. If there is an available frame on the pool, then the function will mark the first available frame encountered as used and return the number of this frame. Otherwise, it will return 0, indicating that there is no available frame currently on the pool.

release_frame

```
static void release_frame(unsigned long _frame_no);
```

The *release_frame* function receives a frame number as argument and then mark this frame as unused. Due to the fact that part of the memory should be inaccessible, a first check is performed to make sure that the frame to be released does not belong to the protected area. Furthermore, since this is a static function, it first needs to identify which one of the pools has this frame, and then access the corresponding bitmap to set the proper bit as 0.

mark_inaccessible

```
void mark_inaccessible(unsigned long _base_frame_no, unsigned long _nframes);
```

The *mark_inaccessible* function defines one region of the memory to be inaccessible from the pools. This is accomplished setting all the bits within this region as used, this mechanism, in conjunction with the check in the *release_frame* function, guarantees that this area of the memory would not have any frames allocated nor released by the other functions.