Resolucion

En primer lugar, para el array que registra el uso de memoria funcionara como un "diccionario", donde el id de la tarea sera la clave y dentro tendra un struct con variables que nos permitiran llevar la cuenta de cuanta memoria se pidio (hasta 4mb) para saber si se excedio o no. Tambien necesitamos un diccionario para llevar cuenta de los mallocos, donde la clave sera el ptr (donde empieza la memoria a reservar), el contenido sera cuanta y si se debe liberar o no.

```
uint32_t MAX_MEMORY = 4MB

typedef struct {
  vaddr_t next_free_memory = 0xA10C0000; //se inicializan con estos valores
  uint32_t memory_used = 0; //este numero esta en bytes
  uin32_t contador_de_mallocos = 0; //cuenta la cantidad de mallocos
  //(ASUMO QUE TODOS LOS ARRAYS TIENEN ESPACIO SUFICIENTE)
  vaddr_t start_mallocos[] //todos los inicios de los mallocos
  uint32_t mallocos_size[] //todos los tamaños de los mallocos
  uint32_t clean[]; //indica si cada malloco debe ser liberado o no
} mmu_malloco_unit;
```
.
```
static mmu_malloc_unit task_memomory_usage[MAX_TASKS] //cada i corresponde con una tarea. de manera 1 a 1 con las del array sched tasks del scheduler
---
Una vez tenemos nuestro array toca registrar nuestras nuevas sycalls en la idt, asi que hay que modificar idt.c y en idt_init() registrarlas con numeros no reservados por intel, ni tampoco que interfieran con el taller.
```c
IDT_ENTRY3(90) //malloco
IDT_ENTRY3(91) //chau
```

Una vez registradas en la idt, (cuando se cargen al iniciar el kernel). Toca definir sus rutinas de atencion y sus handlers para que cumplan lo pedido. Para eso vamos a necesitar que nos pasen el parametro, podriamos pushearlo a la pila de nivel 3 e ir a buscarlo pero en este caso por simplicidad vamos a establecer que el parametro es pasado por el registro EAX. En isr.asm debemos:

```
global _isr90

_isr90:
  pushad
  push eax ;pasamos el size como parametro a la func de c.
  call malloco
  add esp, 4
  mov [ESP+offset_EAX], eax ;hago mov a la pila para no pisarlo con el popad
  ;en eax tengo el puntero que devolvio malloco
```
```

```
  popad
  iret
```

Cuando entro en el handler antes de asignar la memoria debo asegurarme de que no se supero el limite esto lo vamos a poder saber ya que vamos a mantener actualizado el estado de la variable memory used en cada

```
void* malloco(size_t size){
    //la variable global current task almacena el id de la tarea corriendo
    //max_memory tiene el limite actual de 4mb de memoria
    if(task_memory_usage[current_task].memory_used > MAX_MEMORY){
        return NULL;
    } //se supero el limite de memoria para alocar
    else if((task_memory_usage[current_task] + size) > MAX_MEMORY){
        return NULL; //almacenar esto superaria el limite de memoria disponible
    }
    vaddr_t virt = task_memory_usage[current_task].next_free_memory;
    //actualizo el puntero de donde arranca mi proximo cacho de memoria virt
    task_memory_usage[current_task].next_free_memory =+ size;
    //actualizo el uso total
    task_memory_usage[current_task].memory_used =+ size;
    //guardo donde arranca el malloco
    task_memory_usage[current_task].start_mallocos[contador_mallocos] = virt;
    //guardo cuanto ocupo dicho malloco
    task_memory_usage[current_task].size_mallocos[contador_mallocos] =+ size;
    //indico que no debe ser purgado
    task_memory_usage[current_task].clean[contador_mallocos] = 0;
    //actualizo el contador de mallocos
    task_memory_usage[current_task].contador_mallocos++;
    return virt; //devuelve la direccion virtual
}
```

Ahora como malloco es lazy, debo de alguna manera asegurarme que si alguna tarea esta esperando leer o escribir una pagina (que piensa que ya reservo) debo atender ese problemita y donde mejor que en el page fault handler... Asi que ahi dentro debo chequear si lo que genero el page fault me correspondia mapearlo a mi, y efectivamente mapearlo a una fisica e inicializarlo.

Cuando entro a la excepcion page fault (isr 14) estoy en la tarea para la cual tengo que mapearlo y hacerme cargo de mi lazyness. Tambien el handler ya tiene la direccion virtual que tiene que mapear (fue la que genero la int)

```
// devuelve true si se atendió el page fault y puede continuar la ejecució
// y false si no se pudo atender
bool page_fault_handler(vaddr_t virt) {
  print("Atendiendo page fault...", 0, 0, C_FG_WHITE | C_BG_BLACK);
  // Chequeemos si el acceso fue dentro del area on-demand
  if(virt >= ON_DEMAND_MEM_START_VIRTUAL && virt <= ON_DEMAND_MEM_END_VIRTUAL){ //
En caso de que si, mapear la pagina
    uint32_t cr3 = rcr3();
    mmu_map_page(cr3, virt, ON_DEMAND_MEM_START_VIRTUAL, MMU_P | MMU_W | MMU_U);
```

```
// RW usuario
    return true;
  }
  //si el page fault ocurrio en el rango del area de malloco
  if(virt >= 0xA10C0000 && virt <= 0xA10C0000 + MAX_MEMORY){
    uint32_t cr3 = rcr3();
    paddr_t page = next_free_user_page(); //la consigna dice que salen de aca
    zero_page(page); //limpio la pagina
    mmu_map_page(cr3, virt, page, MMU_P | MMU_W | MMU_U); // RW usuario
    //mapeo solo una pagina como dice el enunciado, si mas tarde me vuelven a
llamar mapeo la otra y asi
    return true;
  }
  //si llego hasta aca, la tarea esta leyendo direc q no le corresponde. La apago
    sched_disable_task(current_task);
    //marco toda la memoria para que sea liberada (es un chau pero para todo)
    for(int i = 0; i < task_memory_usage[current_task].contador_de_mallocos;i++){
        task_memory_usage[current_task].clean[i] = 1;
        return;
    }
    sched_next_task(); //llamo al sched para que cambie de tarea (esta ya no es
mas candidata)

}
```

Para poder marcar como en desuso la tarea para que la agarre el garbage, debemos prender la flag del malloco correspondiente. y luego esperar a los 100ticks para que finalmente suceda. Esto lo vamos a hacer recorriendo el array de direcciones de inicio de los mallocos, cuando coincida la que queremos limpiar tendremos el i para prender la flag de clean.

Para implementar chau repetimos el proceso anterior .

```
global _isr91

_isr91:
  pushad
  push eax ;pasaamos el ptr como parametro
  call chau
  add esp, 4
  popad
  iret
```

Lo que debe hacer este puntero es cambiar la flag de la memoria del malloc correspondiente de la tarea como para liberar

```
void chau(void* ptr){
    for(int i = 0; i < task_memory_usage[current_task].contador_de_mallocos;i++){
        if(ptr == task_memory_usage[current_task].start_mallocos[i]){
            task_memory_usage[current_task].clean[i] = 1; //pido que la limpien
```

```
            return;
        }
    }
}
//Para asegurarme la coherencia, cuando la tarea libere la memoria voy a sacar la
direccion por 0
```
.
La tarea arrancaria desactivada en nuestro scheduler, y agregariamos un contador de ticks global para que cuando llegue a 100 interrupciones de clock hacemos un call al garbage collector y reiniciamos el contador otra vez a 0.

Ahora toca definir la tarea, la agregariamos a nuestro sistema como estan las de pong, snake, etc. y escribimos su codigo que va a iterar por todos los clean de todas las tareas, chequeando cuales tienen la flag prendida y desmapeando

```c
void garbageCollector(void) {
    while (true) {
        for(int i = 0; i < MAX_TASKS){
            for(j = 0; j < task_memory_usage[i].contador_de_mallocos;j++){
                if(task_memory_usage[i].clean[j] == 1){ //si me pidieron que lo
limpie
                    //limpio la direc por coherencia (total se va a sobreescribir)
                    task_memory_usage[i].start_mallocos[j] = 0;
                    virt = task_memory_usage[i].start_mallocos[j];
                    get_cr3(current_task);
                    for(int x = 0; x <
task_memory_usage[i].malloco_size[j]/PAGE_SIZE;x++){
                        //calculo cuantas paginas asigno y las desmapeo
                        mmu_unmap_page(cr3, virt);
                        virt =+ PAGE_SIZE * x
                    }
                    task_memory_usage[i].clean[j] == 0;//aviso que limpie
                    task_memory_usage[i].contador_de_mallocos--; //resto la ctad
de mallocos
                    //no actualizo los otros estados ya que no debo reciclar la
memoria
                }


            }
             //resto la ctad total de mallocos en latarea
        }
    }
}

//para conseguir el cr3 de cada tarea entro a su tss
uint32_t get_cr3(uint16_t selector){
    uint16_t idx = selector >> 3;
    tss_t* tss = gdt[idx].base; //aca en realidad hay que reconstruir la direccion
usando ors y las distintas partes de la base pero asumo que asi se entiende
```

```
    return tss->cr3;
}
```

En resumen, la syscall malloco la llama una tarea cuando necesita memoria. Malloco, se encarga de actualizar estados en una estructura y en ella guarda la cantidad de memoria disponible, las direcciones de los siguientes mallocos, si cada malloco se debe purgar o no. Pero no mapea las paginas pedidas, de eso se encarga el page fault handler ya que cuando una tarea que pidio memoria a malloco intenta escribir en alguna de las paginas que pidio, caera en un page fault (y por la guarda que agregamos) si corresponde le damos su pagina, sino la apagamos y asi por cada reserva que nos haga.

Para chau, interactuamos con las variables de estado ya mencionadas y activamos la flag de que un malloc en especifico sea purgado. Para cuando el contador de 100ticks llegue al limite se active el garbacollector y este se encarge de desmapear todas las paginas entregadas.

Correcciones: A continuación te dejo mis correcciones. Quedo a disposición por cualquier consulta o aclaración que necesites.

Corrección segundo parcial Falta un poco de detalle en algunos puntos de la implementación pero la idea está muy completa y no veo errores conceptuales

void* malloco(size_t* size) Construye un array que vive en el espacio de memoria del kernel Los elementos del array son structs con atributos similiares a: task_id, base_virtual_address, size, to_be_freed Crea una entrada de nivel 3 en la IDT La función obtiene la tarea de TR (o algún otro mecanismo equivalente) La función recorre la tabla y acumula los size de los elementos que le correspondan a la tarea (o algún otro mecanismo equivalente) Si no se pasa de los 4 MB incluyendo el size pasado por parámetro, crea un registro nuevo en la tabla Para eso busca una dirección virtual mayor a las del último bloque reservado por la tarea, que permita la definición de páginas contiguas para el size pedido En caso de no existir una dirección virtual que cumpla con las condiciones, devuelve NULL Page Fault Handler Usa la direccion virtual que provocó la excepción de CR2 y la tarea del TR Recorre la tabla y por cada elemento que correspoda a la tarea evalua si la dir virtual pertenece o no al bloque definido por base_virtual_address y size Si ocurre un page fault dentro del área pedible por malloco, hace el mappeo aunque la dirección puntual no haya sido pedida aún (debería fallar). Si está dentro de algún bloque, procede a mapear la página que corresponde a la dirección accedida y luego a inicializarla en cero Idem anterior Si no está dentro de ningún bloque, la tarea se elimina del scheduler, se marcan to_be_freed todos los elementos que corresponden a ella, y se salta a la próxima tarea. No salta a la próxima tarea (falta devolver bool correcto en page_fault_handler). En ningún otro momento del parcial muestra como hacer el salto a otra tarea. void chau(void* ptr) Crea una entrada de nivel 3 en la IDT Obtiene la tarea de TR (o mecanismo equivalente) Recorre la tabla en busca de aquella que corresponda a la tarea y tenga base_virtual_address == ptr y se setea to_be_freed Tarea especial Define una entrada en la gdt para la tss de esta tarea Recorre la tabla en un loop infinito y desmappea todas las páginas que pertenezcan a bloques que tengan to_be_freed seteado Ojo que el orden de estas lineas debería estar invertido:

task_memory_usage[i].start_mallocos[j] = 0; virt = task_memory_usage[i].start_mallocos[j]; Scheduler Falta detalle implementativo pero la idea es clara.

Modifica sched_next_task para que lleve cuenta de los ticks del reloj y retorne el task_id de la tarea especial cada 100 ticks Cuenta que lo haría pero no muestra como. En general se entiende que lo sabe hacer por el resto del parcial. Define una constante con el id de la tarea especial para poder identificarla en el scheduler

Nunca aclara la diferencia entre las tareas de usuario y esta (que es nivel 0, como lograr que sea nivel 0 (config de tss)) Define una variable para contar los 100 clocks Cuenta que lo haría, no cómo lo haría