Universidade Federal de Santa Catarina
Centro Tecnológico
Departamento de Informática e Estatística
INE5424 - Sistemas Operacionais II
Professor: Antônio Augusto Medeiros Fröhlich
Estagiário de Docência: Mateus Krepsky Ludwich
Grupo: 01
Alunos: André Beims Bräscher matrícula: 12106259
Maria Eloísa Costa matrícula: 10200630

P1: Built-in architecture

Operating System Software Architecture

Software Architecture is one of the last understood concepts in computer science. Architecure, as generally defined, goes mostly unquestioned on our minds and is bound to the design of buildings and other physical structures. When brought in context for hardware, Computer Architecture still goes easily on most of us. However, when the subject approaches software, the concept of Software Architecture often goes too abstract on an already abstract world.

Perhaps no one has reasoned on *Operating System Software Architecture* as deeply as Prof. Schr�der-Preikschat. On his book The Logical Design of Parallel Operating Systems and on a series of publications afterwards, he emphatically makes the point on the separation of structural and behavioral properties of operating systems, showing that the same behavior---and indeed the same source code---can be brought about on a variety of architectures. In other words, an operating system can deliver its functionality taking the software architecture that better matches the requirements of the applications running on top of it.

The most known operating system architectures are:

- **Library**: the least structured architecture an OS can assume. OS functionality is implemented and packed as a library to be reused on demand by applications. The software architecture imposed on the application will actually also dictate the OS architecture. The didactic version of OpenEPOS you have been using until now is a good example of this architecture.
- **Monolithic Kernel**: a monolithic-kernel operating system implements all the functionality delivered to applications in the kernel space (i.e. running in supervisor mode, isolated from applications). The main weakness of this architecture, that is, having to handle all possible OS services inside the kernel, is usually overcome by loadable modules. In this way, the monolithic kernel can come up much like a microkernel and extend its functionality on demand by loading modules as new services are requested by applications. Many contemporary OS adhere to this architecture, including Linux and BSD. The major motivation behind this architecture is performance.
- **Microkernel**: an ideal microkernel implements the minimum OS functionality that is necessary to orderly implement the remaining functionality outside the kernel, at user space. Typically, this includes memory management, process management, inter-process communication, and I/O exporting (I/O itself is handled at user space by specific servers, not by kernel device drivers). The imposition of a system call barrier between applications, servers, and the kernel adds considerable overhead and is responsible for the low performance associated with this architecture. In order to overcome this, many microkernels incorporate additional functionality, often making them indistinguishable from monolithic kernels. This,

in turn, brought about such buzz words as *nanokernel* and *picokernel*. Good examples of this architecture are Mach, Amoeba, and L4 .

- **Exokernel**: some times taken as the leanest microkernels, exokernels are actually quite bound to MIT's Parallel and Distributed Operating Systems group. Instead of implementing the minimal functionality needed to implement OS services at user space, exokernel are limited to safely multiplex resources to be used at user space. Since exokernels expose the bare hardware and not a higher level API, porting them to multiple architectures is very hard. Reusing OS abstractions implemented for an exokernel for a given hardware platform on another platform is also hard.
- **Hypervisor**: although hypervisors are usually not considered a particular kind of operating system, but their services and the way such services are made available puts them on a superset of microkernel and exokernel. Policies such as memory and CPU power partitioning are implemented on hypervisors, so in this sense they are more like microkernels than exokernels, but the API delivered to VMs are more like that of an architecture-independent exokernel (if such a thing could exist).

Besides these architectures, EPOS features a *built-in* architecture that consists basically in separating OS code from application code without installing a system call interface. It is a mostly uninteresting architecture, but it is quite useful in two development scenarios:

1. **Application debugging**: even if a system is conceived as single-task, application development is likely to incur in errors that escape from application into the OS (library code). By activating a MMU and marking OS memory as protected, one can more easily catch such bugs (since they will now cause traps that can precisely report the problem when and where it happened).
2. **OS development**: sometimes developing new OS functionality on a library-based architecture can be quite convenient in terms of compilation time.

To do

On your path to make OpenEPOS a real microkernel, the built-in architecture will be very handy. It will enable you to isolate the OS from the application without raising a trap on each small step. The techniques explored while solving the multiple, specialized heaps exercise will also be very helpful here, since isolating application and OS heaps can be achieved using them.

Implement the *built-in* architecture and ensure both code and data from the operating system, even dynamically allocated data, go to the highest addresses, while application's go to the lowest. Traces will be fundamental to demonstrate such an isolation, and the final mapping of OS memory as supervisor mode will confirm it.

Os principais critérios de avaliação para a atividade são:

- Explicar em detalhes o processo de compilação e de ligação das duas arquiteturas de software desenvolvidas até o momento, *library* e *built-in*, ilustrando a explicação com trechos dos respectivos *makefiles*, das ferramentas eposcc e eposmkbi, e do SETUP.
- Definir (e justificar) um mecanismo para identificar o *entry-point* da primeira Thread (i.e. main).
- Modelar e implementar uma primeira versão da classe Task para atuar principalmente como um container para os objetos implicitamente criados pelo SETUP correspondentes a primeira task (uma espécie de Master Task) e também a realação desta classe com a classe Thread.
- Modelar e implementar novos construtores para a classe Thread de forma a permitir a criação de threads sobre tasks específicas.

- Modificar (e justificar) o método init() da classe Thread para usar os construtores de Thread específicos para a primeira task criada pelo SETUP.

Done

- *Explicar em detalhes o processo de compilação e de ligação das duas arquiteturas de software desenvolvidas até o momento, library e built-in, ilustrando a explicação com trechos dos respectivos makefiles, das ferramentas eposcc e eposmkbi, e do SETUP.*

Modo Built-in x Modo Library

Os arquivos gerados para análise podem ser visualizados em library.out e builtin.out

```
make --print-directory run1
make --print-directory all1
```

epos

```
make[1]: Entering directory `/home/maria.eloisa/epos'
```

epos/etc → Define as informações de include/system/config.h de acordo com conteúdo de eposcc.conf e eposmkbi.conf pela execução do makefile da pasta

```
(cd etc && make --print-directory)
make[2]: Entering directory `/home/maria.eloisa/epos/etc'
sed       -e       's/^#define       MODE.*$/#define       MODE       builtin/'       -i
/home/maria.eloisa/epos/include/system/config.h
sed       -e       's/^#define       MODE.*$/#define       MODE       library/'       -i
/home/maria.eloisa/epos/include/system/config.h
sed       -e       's/^#define       ARCH.*$/#define       ARCH       ia32/'       -i
/home/maria.eloisa/epos/include/system/config.h
sed       -e       's/^#define       MACH.*$/#define       MACH       pc/'       -i
/home/maria.eloisa/epos/include/system/config.h
sed       -e       's/^#define       MMOD.*$/#define       MMOD       legacy/'       -i
/home/maria.eloisa/epos/include/system/config.h
sed       -e       's/^#define       APPL.*$/#define       APPL       task_test/'       -i
/home/maria.eloisa/epos/include/system/config.h
sed       -e       's/^#define       __mode_.*__$/#define       __mode_builtin__/'       -i
/home/maria.eloisa/epos/include/system/config.h
sed       -e       's/^#define       __mode_.*__$/#define       __mode_library__/'       -i
/home/maria.eloisa/epos/include/system/config.h
sed       -e       's/^#define       __arch_.*__$/#define       __arch_ia32__/'       -i
/home/maria.eloisa/epos/include/system/config.h
sed       -e       's/^#define       __mach_.*__$/#define       __mach_pc__/'       -i
/home/maria.eloisa/epos/include/system/config.h
sed       -e       's/^#define       __mmod_.*__$/#define       __mmod_legacy__/'       -i
/home/maria.eloisa/epos/include/system/config.h
make[2]: Leaving directory `/home/maria.eloisa/epos/etc'
```

epos/tools

```
(cd tools && make --print-directory)
make[2]: Entering directory `/home/maria.eloisa/epos/tools'
```

epos/tools/eposcc → Define o path pro EPOS de acordo com o que está definido no arquivo bin/eposcc

EPOS=/home/maria.eloisa/epos

```
(cd eposcc && make --print-directory)
make[3]: Entering directory `/home/maria.eloisa/epos/tools/eposcc'
install -m 775 eposcc /home/maria.eloisa/epos/bin
sed          -e          's/^EPOS=.*$/EPOS=\/home\/maria.eloisa\/epos/'          -i
/home/maria.eloisa/epos/bin/eposcc
make[3]: Leaving directory `/home/maria.eloisa/epos/tools/eposcc'
```

epos/tools/eposmkbi → Instala o eposmkbi dentro da pasta bin

```
(cd eposmkbi && make --print-directory)
make[3]: Entering directory `/home/maria.eloisa/epos/tools/eposmkbi'
install -m 775 eposmkbi /home/maria.eloisa/epos/bin
make[3]: Leaving directory `/home/maria.eloisa/epos/tools/eposmkbi'
make[2]: Leaving directory `/home/maria.eloisa/epos/tools'
```

epos/src

```
(cd src && make --print-directory)
make[2]: Entering directory `/home/maria.eloisa/epos/src'
```

epos/src/utility

```
(cd utility && make --print-directory)
make[3]: Entering directory `/home/maria.eloisa/epos/src/utility'
make[3]: Nothing to be done for `all'.
make[3]: Leaving directory `/home/maria.eloisa/epos/src/utility'
```

epos/src/architecture

```
(cd architecture && make --print-directory)
make[3]: Entering directory `/home/maria.eloisa/epos/src/architecture'
```

epos/src/architecture/ia32 → Definição das bibliotecas internas crt do epos são inseridas na pasta epos/lib

```
crts:       $(CRTS)
            $(INSTALL) $^ $(LIB)
```

```
(cd ia32 && make --print-directory)
make[4]: Entering directory `/home/maria.eloisa/epos/src/architecture/ia32'
install ia32_crt0.o ia32_crtbegin.o ia32_crtend.o /home/maria.eloisa/epos/lib
make[4]: Leaving directory `/home/maria.eloisa/epos/src/architecture/ia32'
make[3]: Leaving directory `/home/maria.eloisa/epos/src/architecture'
```

epos/src/machine

```
(cd machine && make --print-directory)
make[3]: Entering directory `/home/maria.eloisa/epos/src/machine'
```

epos/src/machine/common

```
(cd common && make --print-directory)
make[4]: Entering directory `/home/maria.eloisa/epos/src/machine/common'
make[4]: Nothing to be done for `all'.
make[4]: Leaving directory `/home/maria.eloisa/epos/src/machine/common'
```

epos/src/machine/pc

```
(cd pc && make --print-directory)
make[4]: Entering directory `/home/maria.eloisa/epos/src/machine/pc'
make[4]: Nothing to be done for `all'.
make[4]: Leaving directory `/home/maria.eloisa/epos/src/machine/pc'
make[3]: Leaving directory `/home/maria.eloisa/epos/src/machine'
```

epos/src/abstraction

```
(cd abstraction && make --print-directory)
make[3]: Entering directory `/home/maria.eloisa/epos/src/abstraction'
make[3]: Nothing to be done for `all'.
make[3]: Leaving directory `/home/maria.eloisa/epos/src/abstraction'
```

epos/src/setup → Define o pc_setup para a imagem do epos

```
$(MACH)_setup:      $(MACH)_setup.o
             $(LD) $(LDFLAGS) -L$(CCLIB) --omagic --section-start .init=$(SETUP_ADDR)
-o $@ $^ -l$(LINIT) -l$(LMACH) -l$(LARCH) -l$(LUTIL) -lgcc

(cd setup && make --print-directory)
make[3]: Entering directory `/home/maria.eloisa/epos/src/setup'
install pc_setup /home/maria.eloisa/epos/img
make[3]: Leaving directory `/home/maria.eloisa/epos/src/setup'
```

epos/src/boot → Define o arquivo de boot, criando um temporário, passando as informações para img/pc_boot e removendo o arquivo temporário

```
(cd boot && make --print-directory)
make[3]: Entering directory `/home/maria.eloisa/epos/src/boot'
dd if=pc_boot of=pc_boot.tmp ibs=32 skip=1 obs=512 >& /dev/null
install pc_boot.tmp /home/maria.eloisa/epos/img/pc_boot
rm -f pc_boot.tmp
make[3]: Leaving directory `/home/maria.eloisa/epos/src/boot'
```

epos/src/system

```
(cd system && make --print-directory)
make[3]: Entering directory `/home/maria.eloisa/epos/src/system'
/usr/local/ia32/gcc-4.4.4/bin/ia32-ld     -nostdlib   -L/home/maria.eloisa/epos/lib   -
Bstatic  -L`/usr/local/ia32/gcc-4.4.4/bin/ia32-gcc  -ansi  -c    -Wa,--32  -print-file-
name=` --nmagic \
                --section-start .init=0xff700000 \
                --section-start .ctors=0xff740000 \
                --entry=_init -o pc_system \
                /home/maria.eloisa/epos/lib/ia32_crtbegin.o \
                system_scaffold.o \
                /home/maria.eloisa/epos/lib/ia32_crtend.o \
                --whole-archive \
                -lsys_ia32 -lmach_ia32 -larch_ia32 \
                --no-whole-archive \
                -lutil_ia32 -lgcc
install application_scaffold.o pc_application.o
/usr/local/ia32/gcc-4.4.4/bin/ia32-objcopy -L _end pc_system
install pc_system /home/maria.eloisa/epos/img
install pc_application.o /home/maria.eloisa/epos/lib
```

```
ld para o modo bultin

$(MACH)_system_builtin: $(OBJS)
            $(LD) $(LDFLAGS) -L$(CCLIB) --nmagic \
            --section-start $(MACH_CODE_NAME)=$(SYS_CODE_ADDR) \
            --section-start $(MACH_DATA_NAME)=$(SYS_DATA_ADDR) \
            --entry=_init -o $(MACH)_system \
            $(LIB)/$(ARCH)_crtbegin.o \
            system_scaffold.o \
            $(LIB)/$(ARCH)_crtend.o \
            --whole-archive \
            -l$(LSYS) -l$(LMACH) -l$(LARCH) \
            --no-whole-archive \
            -l$(LUTIL) -lgcc
            $(INSTALL) application_scaffold.o $(MACH)_application.o

/usr/local/ia32/gcc-4.4.4/bin/ia32-ld    -nostdlib  -L/home/maria.eloisa/epos/lib  -
Bstatic -i system_scaffold.o -o pc_system.o
install application_scaffold.o pc_application.o
install pc_system.o /home/maria.eloisa/epos/lib
install pc_application.o /home/maria.eloisa/epos/lib

ld para o modo Library

$(MACH)_system_library: system_scaffold.o application_scaffold.o
            $(LD) $(LDFLAGS) -i system_scaffold.o -o $(MACH)_system.o
            $(INSTALL) application_scaffold.o $(MACH)_application.o

O comando acima usa o modo incremental do ld para rodar em modo LIBRARY

make[3]: Leaving directory `/home/maria.eloisa/epos/src/system'
```

epos/src/init

```
(cd init && make --print-directory)
make[3]: Entering directory `/home/maria.eloisa/epos/src/init'
/usr/local/ia32/gcc-4.4.4/bin/ia32-ld    -nostdlib  -L/home/maria.eloisa/epos/lib  -
Bstatic  -L`/usr/local/ia32/gcc-4.4.4/bin/ia32-gcc  -ansi  -c   -Wa,--32  -print-file-
name=` --omagic \
               --section-start .init=0x00200000 \
               --entry=_init -o pc_init \
               /home/maria.eloisa/epos/lib/ia32_crtbegin.o \
               init_first.o init_system.o \
               /home/maria.eloisa/epos/lib/ia32_crtend.o \
               -linit_ia32 \
               -R /home/maria.eloisa/epos/src/system/pc_system \
               -lutil_ia32 -lgcc
install init_application.o pc_init_application.o
install pc_init /home/maria.eloisa/epos/img
install pc_init_application.o /home/maria.eloisa/epos/lib
install init_first.o pc_init_first.o
install init_system.o pc_init_system.o
install init_application.o pc_init_application.o
install pc_init_first.o /home/maria.eloisa/epos/lib
```

```
install pc_init_system.o /home/maria.eloisa/epos/lib
install pc_init_application.o /home/maria.eloisa/epos/lib
make[3]: Leaving directory `/home/maria.eloisa/epos/src/init'
make[2]: Leaving directory `/home/maria.eloisa/epos/src'
```

epos/src/app → Gera a aplicação na pasta epos/img

```
(cd app && make --print-directory)
make[2]: Entering directory `/home/maria.eloisa/epos/app'
/home/maria.eloisa/epos/bin/eposcc --builtin  --gc-sections  -o task_test task_test.o
/home/maria.eloisa/epos/bin/eposcc --library  --gc-sections  -o task_test task_test.o
install task_test /home/maria.eloisa/epos/img
make[2]: Leaving directory `/home/maria.eloisa/epos/app'
```

epos/img → Faz a criação da imagem do epos de acordo com as especificações feitas durante o processo de compilação e ligação

```
(cd img && make --print-directory)
make[2]: Entering directory `/home/maria.eloisa/epos/img'
/home/maria.eloisa/epos/bin/eposmkbi /home/maria.eloisa/epos task_test.img task_test

EPOS bootable image tool

  EPOS mode: builtin
  EPOS mode: library
  Machine: pc
  Model: legacy
  Processor: ia32 (32 bits, little-endian)
  Memory: 262144 KBytes
  Boot Length: 128 - 512 (min - max) KBytes
  Node id: will get from the network

  Creating EPOS bootable image in "task_test.img":
    Adding boot strap "/home/maria.eloisa/epos/img/pc_boot": done.
    Adding setup "/home/maria.eloisa/epos/img/pc_setup": done.
    Adding init "/home/maria.eloisa/epos/img/pc_init": done.
    Adding system "/home/maria.eloisa/epos/img/pc_system": done.
    Adding application "task_test": done.
    Adding system info: done.

  Adding specific boot features of "pc": done.

  Image successfully generated (75916 bytes)!

make[2]: Leaving directory `/home/maria.eloisa/epos/img'
```

epos/img → Carregamento do SO para inicialização do sistema

Fica nítida a separação entre library e built-in do SO e a Aplicação pela parte verde e laranja do sistema

```
(cd img && make --print-directory run)
make[2]: Entering directory `/home/maria.eloisa/epos/img'
qemu-system-i386 -smp 1 -m 262144k -nographic -no-reboot -fda  task_test.img | tee
task_test.out
Setting up this machine as follows:
  Processor:    IA32 at 2205 MHz (BUS clock = 125 MHz)
  Memory:       262144 Kbytes [0x00000000:0x10000000]
```

```
  User memory:  261752 Kbytes [0x00000000:0x0ff9e000]
  PCI aperture: 45064 Kbytes [0xfc000000:0xfec02000]
  Node Id:      will get from the network!
  Setup:        18944 bytes
  Init:         6448 bytes
  OS code:      43216 bytes     data: 640 bytes stack: 16384 bytes
  APP code:     4192 bytes      data: 16797696 bytes
  APP code:     27568 bytes     data: 512 bytes
```

- *Definir (e justificar) um mecanismo para identificar o entry-point da primeira Thread (i.e. main).*

Inicialmente teríamos que realizar algumas alterações para que o sistema iniciasse em modo built-in e não mais em modo library, como ocorria desde o início do semestre. Para tanto, tornou-se necessário fazer uma primeira alteração em traits.h

include/system/traits.h

```
template<> struct Traits<Build>
{
    enum {LIBRARY, BUILTIN};
    static const unsigned int MODE = BUILTIN;
    ...
```

Como o código disponibilizado para teste, *task_test.cc*, ainda não era compilável por falta da classe Task que não existe ainda no sistema, o comando *nm -C -n app/segment_test* foi executado inicialmente para que pudéssemos iniciar a análise de como o sistema estava executando e o que precisaria ser feito. Os dados foram salvos em *segment_test.out.*

Ao tentar rodar o código da aplicação de testes do segmento, encontramos um erro. No modo built-in é possível termos várias aplicações rodando e, faz-se necessário identificar onde na memória fica a main da aplicação. A chamada para o método main é apenas um endereço na área de código.

```
      make[2]: Leaving directory `/home/maria.eloisa/epos/img'
      (cd img && make --print-directory run)
      make[2]: Entering directory `/home/maria.eloisa/epos/img'
      qemu-system-i386 -smp 1 -m 262144k -nographic -no-reboot -fda  segment_test.img
| tee segment_test.out
      Setting up this machine as follows:
        Processor:    IA32 at 2205 MHz (BUS clock = 125 MHz)
        Memory:       262144 Kbytes [0x00000000:0x10000000]
        User memory:  261752 Kbytes [0x00000000:0x0ff9e000]
        PCI aperture: 45064 Kbytes [0xfc000000:0xfec02000]
        Node Id:      will get from the network!
        Setup:        18944 bytes
        Init:         5104 bytes
        OS code:      41424 bytes     data: 608 bytes     stack: 16384 bytes
        APP code:     2176 bytesdata: 16797696 bytes
      [024;073HPANIC!make[2]: Leaving directory `/home/maria.eloisa/epos/img'
      make[1]: Leaving directory `/home/maria.eloisa/epos'
```

Com a ajuda de outro grupo conseguimos entender qual era o problema que estava acontecendo. Mesmo ativando todos os debugs, o sistema não denunciava qual era o problema que impedia o EPOS de rodar. Quando o EPOS está em modo LIBRARY, ele ajusta o ponteiro de entrada da aplicação para __epos_app_entry em thread_init.cc, que fará a chamada de main. No caso, como alteramos para built-in, entry está ainda apontando para __epos_app_entry, porém não há informação nenhuma.

É necessário fazer uma alteração para que entry ainda consiga apontar para main. Com isso é adicionada a seguinte linha:

src/abstraction/thread_init.cc

```
int (* entry)();
if(Traits<Build>::MODE == Traits<Build>::LIBRARY)
    entry = reinterpret_cast<int (*)()>(__epos_app_entry);
else
    entry = reinterpret_cast<int (*)()>(System::info()->lm.app_entry);
```

src/setup/pc_setup.cc

```
si->lm.app_entry = app_elf->entry();
```

Enquanto o sistema está rodando o setup, durante a leitura da imagem do epos, é gerado um system_info, com todas as informações necessárias para executar a aplicação. É necessário então saber onde está o segmento de código da main para que o sistema possa identificar como continuar a executar. *lm* é o Load_Map que contém o endereço da aplicação. *app_entry* receberá o valor o valor de entrada do elf da aplicação que indica por onde o sistema deve executar.

- *Modelar e implementar uma primeira versão da classe Task para atuar principalmente como um container para os objetos implicitamente criados pelo SETUP correspondentes a primeira task (uma espécie de Master Task) e também a relação desta classe com a classe Thread.*

Coloca-se as classes System e Thread como amigas da classe Task pois elas precisarão ter informações de quais Tasks estão rodando no sistema (system) e a qual Task a thread pertence, por exemplo.

```
#ifndef __task_h
#define __task_h

#include <address_space.h>
#include <segment.h>
#include <mmu.h>
#include <system.h>
#include <utility/list.h>

__BEGIN_SYS

class Task
{
   friend class System;
   friend class Thread;
```

```
public:
    Task();
```

É criado um construtor de task passando o segmento de código e de dados como parâmetro. É importante que estes sejam constantes pois faz com que não seja permitida nenhuma alteração direta sobre a Task, criada em *task_test.cc* como constante. Quando definido que task é constante, toda chamada internda de task também precisa ser constante.

```
    Task(const Segment & cs, const Segment & ds);
    ~Task();
```

Métodos de retorno de informações da task.

```
    Address_Space * address_space() const;
    Segment * code_segment() const;
    Segment * data_segment() const;
    CPU::Phy_Addr code() const;
    CPU::Phy_Addr data() const;
    static Task * self();
```

Inserção e remoção da thread na lista de controle de threads da task.

```
    void insert(Thread * t);
    void remove(Thread * t);

private:
```

Atributos da task.

```
    Address_Space * _address_space;
    Segment * _code_segment;
    Segment * _data_segment;
    CPU::Phy_Addr _code;
    CPU::Phy_Addr _data;
```

Em uma troca de task, permite que seja feita a ativação do endereço de memória da task que está sendo ativada.

```
    void activate() const { _address_space->activate(); };

    static void init();
```

_threads armazena as threads de uma dada task

```
    static Simple_List<Thread> _threads;
```

_master é a primeira task do sistema

```
    static Task * _master;
};

__END_SYS

#include <thread.h>

#endif
```

Para a inicialização da primeira *task*, (a *_master*), a função *System::init*, ficou da seguinte forma:

src/abstraction/system_init.cc

```
void System::init()
{
       Task::init();

    if(Traits<Alarm>::enabled)
        Alarm::init();
}
```

Com isso, a função em task_init será chamada. Como pode ser visto essa função apenas serve para alocar memória para a primeira task bem como suas variáveis. Considera-se que a primeira *task* é parte do sistema.

src/abstraction/task_init.cc

```
#include <utility/elf.h>
#include <mmu.h>
#include <system.h>
#include <task.h>

__BEGIN_SYS

void Task::init()
{
       System_Info<Machine> * si = System::info();

    _master = new (SYSTEM) Task();
    _master->_address_space = new (SYSTEM) Address_Space (MMU::current());
    _master->_code_segment = new (SYSTEM) Segment (CPU::Phy_Addr(si->lm.app_code),
si->lm.app_code_size);
    _master->_data_segment = new (SYSTEM) Segment (CPU::Phy_Addr(si->lm.app_data),
si->lm.app_data_size);
    _master->_code = CPU::Phy_Addr(Memory_Map<Machine>::APP_CODE);
    _master->_data = CPU::Phy_Addr(Memory_Map<Machine>::APP_DATA);
}

__END_SYS
```

Em init() é gerado um novo address space para a task, através do comando "_address_space = new (SYSTEM) Address_Space (MMU::current());", que pega o endereço do diretório de páginas atual da MMU, que é usado para criar um novo *Address_Space.*

Os segmentos de código e de dados são obtidos a partir do System::info que possui informações do Load_Map para carregar o segmento de código e de dados.

src/abstraction/task.cc

```
#include <system.h>
#include <task.h>

__BEGIN_SYS

Task * Task::_master;
Simple_List<Thread> Task::_threads;

Task::Task() { }

Task::Task(const Segment & cs, const Segment & ds) {
   _address_space = new (SYSTEM) Address_Space (MMU::current());
   _code_segment = const_cast<Segment *>(&cs);
   _data_segment = const_cast<Segment *>(&ds);
   _code = CPU::Phy_Addr(Memory_Map<Machine>::APP_CODE);
   _data = CPU::Phy_Addr(Memory_Map<Machine>::APP_DATA);
}

Task::~Task() {
   delete _address_space;
   delete _code_segment;
   delete _data_segment;

   while(!_threads.empty()) {
       Thread * t = _threads.remove()->object();
       delete t;
   }
}

Task * Task::self() {
       return Thread::self()->_task;
}

void Task::insert(Thread * t) {

   kout << "Task::insert(Thread * t): " << _threads.size() << " == 0" << endl;

   _threads.insert(&t->_link_task);
   t->_task = this;

   kout << "Task::insert(Thread * t): " << _threads.size() << " == 1" << endl;

}

void Task::remove(Thread * t) {

   _threads.remove(&t->_link_task);

}

Address_Space * Task::address_space() const {
   return _address_space;
```

```
}

Segment * Task::code_segment() const {
   return _code_segment;
}

Segment * Task::data_segment() const {
   return _data_segment;
}

CPU::Phy_Addr Task::code() const {
   return _code;
}

CPU::Phy_Addr Task::data() const {
   return _data;
}

__END_SYS
```

No construtor da task, é gerado um novo address space para a task, através do comando "`_address_space = new (SYSTEM) Address_Space (MMU::current()).`" Após isso, obtém-se os segmentos de código e de dados, respectivamente, através dos comandos:

```
_code_segment = const_cast<Segment *>(&cs);
_data_segment = const_cast<Segment *>(&ds);
```

Tais comandos fazem um *const_cast* (para poder fazer a atribuição de valor constante), de ponteiro para segmento. Finalmente são adicionados os endereços físicos do código e dos dados.

Já o destrutor da Task funciona da seguinte maneira:

```
Task::~Task() {
   delete _address_space;
   delete _code_segment;
   delete _data_segment;

   while(!_threads.empty()) {
       Thread * t = _threads.remove()->object();
       delete t;
   }
}
```

Primeiramente é feito o comando delete sobre o espaço de endereçamento, o segmento de código e o segmento de dados. Então basta excluir as *threads* que estão rodando sobre essa *task*, através do laço *while* que fica removendo *threads* da lista e as deletando, até que a lista esteja vazia.

Para fazer a inserção de novas *threads* para rodarem sobre o código da *task*, elaborou-se a seguinte função:

```
void Task::insert(Thread * t) {

   kout << "Task::insert(Thread * t): " << _threads.size() << " == 0" << endl;
```

```
    _threads.insert(&t->_link_task);
    t->_task = this;

    kout << "Task::insert(Thread * t): " << _threads.size() << " == 1" << endl;

}
```

Tal função apenas usa o método da lista para a inserção do *link* e então a *thread* adicionada recebe a referência para a *task*, através de "`t->_task = this;`".

Foi incluído também um sistema para controle. Caso uma thread seja deletada, a referência em *_threads* deve ser removida a partir do método remove da *task.*

```
void Task::remove(Thread * t) {

    _threads.remove(&t->_link_task);

}
```

Para concluir o *task.cc*, foram adicionados mecanismos para acesso às variáveis de *task*, como mostrado a seguir:

```
Address_Space * Task::address_space() const {
    return _address_space;
}

Segment * Task::code_segment() const {
    return _code_segment;
}

Segment * Task::data_segment() const {
    return _data_segment;
}

CPU::Phy_Addr Task::code() const {
    return _code;
}

CPU::Phy_Addr Task::data() const {
    return _data;
}
```

- *Modelar e implementar novos construtores para a classe Thread de forma a permitir a criação de threads sobre tasks específicas.*

Uma vez feitas as alterações de forma a obter uma classe *task*, passou a ser necessário alterar a construção e de deleção de *threads* de forma que estas, respectivamente "se insiram" e "se retirem" da *task* a qual elas pertencem. Assim, novo construtor e destrutor de *thread* ficam da seguinte forma:

include/thread.h

```
class Thread
{
    friend class Init_First;
```

A *Task* passa a ser uma classe amiga de *Thread* para que possa buscar informações sobre suas *threads*.

```
    friend class Task;
    friend class Scheduler<Thread>;
    friend class Synchronizer_Common;
    friend class Alarm;
    friend class IA32;
    ...

public:
    template<typename ... Tn>
    Thread(int (* entry)(Tn ...), Tn ... an);
    template<typename ... Cn, typename ... Tn>
    Thread(const Configuration & conf, int (* entry)(Tn ...), Tn ... an);
```

Foram criados dois novos construtores, para que quando uma *thread* seja criada, ela informe a *task* a qual pertence. Caso essa informação não seja dada na sua construção, a *thread* é criada sobre a *task* corrente.

```
    template<typename ... Tn>
    Thread(Task * task, int (* entry)(Tn ...), Tn ... an);
    template<typename ... Cn, typename ... Tn>
    Thread(Task * task, const Configuration & conf, int (* entry)(Tn ...), Tn ...
an);
    ~Thread();
    ...

protected:
    char * _stack;
    Context * volatile _context;
    volatile State _state;
    Queue * _waiting;
    Thread * volatile _joining;
    Queue::Element _link;
    Simple_List<Thread>::Element _link_task;
    Task * _task;
     ...

template<typename ... Tn>
inline Thread::Thread(int (* entry)(Tn ...), Tn ... an)
: _state(READY), _waiting(0), _joining(0), _link(this, NORMAL), _link_task(this)
{
    lock();
    _stack = new (SYSTEM) char[STACK_SIZE];
    _context = CPU::init_stack(_stack, STACK_SIZE, &implicit_exit, entry, an ...);
    running()->_task->insert(this);
    constructor(entry, STACK_SIZE); // implicit unlock
}
```

```
template<typename ... Cn, typename ... Tn>
inline Thread::Thread(const Configuration & conf, int (* entry)(Tn ...), Tn ... an)
: _state(conf.state), _waiting(0), _joining(0), _link(this, conf.criterion),
_link_task(this)
{
    lock();
    _stack = new (SYSTEM) char[conf.stack_size];
    _context = CPU::init_stack(_stack, conf.stack_size, &implicit_exit, entry, an
...);
    running()->_task->insert(this);
    constructor(entry, conf.stack_size); // implicit unlock
}

template<typename ... Tn>
inline Thread::Thread(Task * task, int (* entry)(Tn ...), Tn ... an)
: _state(READY), _waiting(0), _joining(0), _link(this, NORMAL), _link_task(this),
_task(task)
{
    lock();
    _stack = new (SYSTEM) char[STACK_SIZE];
    _context = CPU::init_stack(_stack, STACK_SIZE, &implicit_exit, entry, an ...);
    _task->insert(this);
    constructor(entry, STACK_SIZE); // implicit unlock
}

template<typename ... Cn, typename ... Tn>
inline Thread::Thread(Task * task, const Configuration & conf, int (* entry)(Tn ...),
Tn ... an)
: _state(conf.state), _waiting(0), _joining(0), _link(this, conf.criterion),
_link_task(this), _task(task)
{
    lock();
    _stack = new (SYSTEM) char[conf.stack_size];
    _context = CPU::init_stack(_stack, conf.stack_size, &implicit_exit, entry, an
...);
    _task->insert(this);
    constructor(entry, conf.stack_size); // implicit unlock
}
```

src/abstraction/thread.cc

```
Thread::~Thread()
{
    lock();

    db<Thread>(TRC) << "~Thread(this=" << this
                    << ",state=" << _state
                    << ",priority=" << _link.rank()
                    << ",stack={b=" << _stack
                    << ",context={b=" << _context
                    << "," << *_context << "})" << endl;

    // The running thread cannot delete itself!
    assert(_state != RUNNING);
```

```
    switch(_state) {
    case RUNNING:  // For switch completion only: the running thread would have
deleted itself! Stack wouldn't have been released!
        exit(-1);
        break;
    case READY:
        _scheduler.remove(this);
        _thread_count--;
        break;
    case SUSPENDED:
        _scheduler.resume(this);
        _scheduler.remove(this);
        _thread_count--;
        break;
    case WAITING:
        _waiting->remove(this);
        _scheduler.resume(this);
        _scheduler.remove(this);
        _thread_count--;
        break;
    case FINISHING: // Already called exit()
        break;
    }

    if(_joining)
        _joining->resume();
```

No destrutor da *thread* é feita a remoção da thread da lista da *task* para que a *task* não fique com informações inválidas quando for destruída.

```
    _task->remove(this);

    unlock();

    delete _stack;
}

void Thread::dispatch(Thread * prev, Thread * next, bool charge)
{
    if(charge) {
        if(Criterion::timed)
            _timer->reset();
    }

    if(prev != next) {
        if(prev->_state == RUNNING)
            prev->_state = READY;
        next->_state = RUNNING;
```

Quando é feita a troca de *threads* é verificado se a *task* a qual a *thread* pertence é a mesma da próxima para que as informações de endereço de memória permaneçam consistentes entre uma troca e outra.

```
         if(prev->_task != next->_task){
            next->_task->activate();
        }
```

```
        db<Thread>(TRC) << "Thread::dispatch(prev=" << prev << ",next=" << next <<
")" << endl;
        db<Thread>(INF) << "prev={" << prev << ",ctx=" << *prev->_context << "}" <<
endl;
        db<Thread>(INF) << "next={" << next << ",ctx=" << *next->_context << "}" <<
endl;


        CPU::switch_context(&prev->_context, next->_context);
    }

    unlock();
}
```

- *Modificar (e justificar) o método init() da classe Thread para usar os construtores de Thread específicos para a primeira task criada pelo SETUP.*

Visto que a primeira *task* do sistema é a *_master*, criada no *system_init*, tanto a *thread main* quanto a *idle* pertencerão a esta *task*. A partir deste momento no sistema toda *thread* pertencerá a uma *task*. As *threads* iniciais do sistema passam a pertencer a primeira *task* do sistema. A *task _master* seria a *task* corrente no sistema.

src/abstraction/thread_init.cc

```
first = new (SYSTEM) Thread(Task::_master, Configuration(RUNNING, MAIN), entry);
    new (SYSTEM) Thread(Task::_master, Configuration(READY, IDLE), &idle);
```

A aplicação *task_test* foi alterada para utilizar também *threads*:

app/task_test.cc

```
// EPOS Task Test Program

#include <utility/ostream.h>
#include <alarm.h>
#include <thread.h>
#include <task.h>

using namespace EPOS;

const int iterations = 10;

int func_a(void);
int func_b(void);

Thread * a;
Thread * b;
Thread * m;

OStream cout;
```

```
int main()
{
    cout << "Task test" << endl;

    m = Thread::self();

    cout << "I'll try to clone myself:" << endl;

    const Task * task0 = Task::self();
    Address_Space * as0 = task0->address_space();
    cout << "My address space's page directory is located at " << as0->pd() << endl;

    const Segment * cs0 = task0->code_segment();
    CPU::Log_Addr code0 = task0->code();
    cout << "My code segment is located at "
         << static_cast<void *>(code0)
         << " and it is " << cs0->size() << " bytes long" << endl;

    const Segment * ds0 = task0->data_segment();
    CPU::Log_Addr data0 = task0->data();
    cout << "My data segment is located at "
         << static_cast<void *>(data0)
         << " and it is " << ds0->size() << " bytes long" << endl;

    cout << "Creating and attaching segments:" << endl;
    Segment cs1(cs0->size());
    CPU::Log_Addr code1 = as0->attach(cs1);
    cout << "  code => " << code1 << " done!" << endl;
    Segment ds1(ds0->size());
    CPU::Log_Addr data1 = as0->attach(ds1);
    cout << "  data => " << data1 << " done!" << endl;

    cout << "Copying segments:";
    memcpy(code1, code0, cs1.size());
    cout << " code => done!" << endl;
    memcpy(data1, data0, ds1.size());
    cout << " data => done!" << endl;

    cout << "Detaching segments:";
    as0->detach(cs1);
    as0->detach(ds1);
    cout << " done!" << endl;

    cout << "Creating a clone task:";
    Task * task1 = new Task(cs1, ds1);
       a = new Thread(task1, &func_a);
    b = new Thread(&func_b);
    cout << " done!" << endl;

       a->join();
       b->join();

    cout << "Deleting the cloned task:";
       delete a;
        delete b;
```

```
    delete task1;
    cout << " done!" << endl;

    cout << "I'm also done, bye!" << endl;

    return 0;
}

int func_a(void) {
       for(int i = iterations; i > 0; i--) {
              cout << "HEARTY HELLO! I'm A!!" << endl;
              Thread::yield();
       }
       return 1;
}

int func_b(void) {
       for(int i = iterations; i > 0; i--) {
              cout << "HELLO MAMA!! I'm B!!" << endl;
              Thread::yield();
       }
       return 1;
}
```