ID: .....

Laboratório de Sistemas Computacionais: Sistemas Operacionais

São José dos Campos - Brasil

Fevereiro de 2018

ID: .....

Laboratório de Sistemas Computacionais: Sistemas Operacionais

Relatório apresentado à Universidade Federal de São Paulo como parte dos requisitos para aprovação na disciplina de Laboratório de Sistemas Computacionais: Sistemas Operacionais.

Docente: Prof. Dr. Tiago de Oliveira

Universidade Federal de São Paulo - UNIFESP

Instituto de Ciência e Tecnologia - Campus São José dos Campos

São José dos Campos - Brasil

Fevereiro de 2018

Resumo

Este projeto descreve a solução de arquitetura, desenvolvimento e teste de um sistema operacional capaz de realizar a execução de programas, troca de contexto - manual e em segundo plano, e com controle de entrada/saída e término de programas - e operações com arquivos. Ele foi criado especificamente para a arquitetura desenvolvida no projeto anterior (1) e foi desenvolvido em uma versão customizada de *assembly* (2).

Palavras-chaves: sistemas operacionais. sistemas operativos. algoritmos de paginação. algoritmos de escalonamento. DMA.

Lista de ilustrações

Sumário

1 Introdução

Sistemas computacionais modernos, segundo (3), consistem em um processador (ou mais), memória principal, discos, teclado, *mouse*, entre outros diversos dispositivos, assim se caracterizando como um sistema complexo.

Quando um programa é rodado, de acordo com (4), temos instruções sendo executadas. Muitos milhões delas, a cada segundo, são obtidas da memória, decodificadas e executadas pelo processador. Para que as instruções cumpram seu propósito, todos os componentes descritos anteriormente devem trabalhar em conjunto, de maneira que os recursos do sistema sejam divididos de forma justa entre os programas.

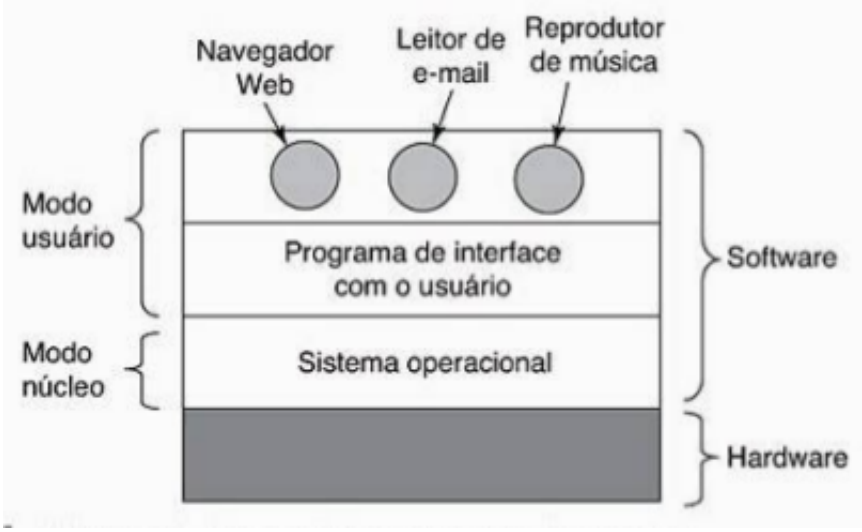
A fim de gerir o funcionamento dos sistemas computacionais, os computadores contam com um dispositivo de software denominado **sistema operacional**, cujo trabalho é, segundo (3), "*fornecer aos programas do usuário um modelo de computador melhor, mais simples e mais limpo, e lidar com o gerenciamento de todos os recursos mecionados*".

Sistemas operacionais estão presentes na maioria dos sistemas computacionais utilizados hoje em dia. Em 2015, segundo (5), os sistemas operacionais móveis já haviam se tornado os mais populares no mundo, equipando mais de 66% dos dispositivos computacionais e superando sistemas operacionais tradicionais como *Windows*, *MacOS* e *GNU/Linux*.

A Figura 1 apresenta onde o sistema operacional se encaixa na estrutura de um sistema computacional. Na parte inferior, em cinza, se apresenta o *hardware*, o qual consiste em todos os componentes físicos do sistema e está no nível mais baixo da estrutura.

Logo acima, temos o sistema operacional. Este opera em modo *kernel*, de maneira que tem acesso completo ao *hardware* e pode executar qualquer operação no processador.

Figura 1 – O sistema operacional em um sistema computacional.



Fonte: Modern Operating Systems (3)

O sistema operacional realiza uma interface entre o *hardware* e o restante do *software*, o qual opera em modo usuário e tem permissão limitada quanto às instruções que pode solicitar do *hardware*.

A arquitetura dos processadores, em nível de linguagem de máquina, costuma ser primitiva e trabalhosa de se programar. A complexidade de um nível tão baixo é abstraída pelo sistema operacional para a elaboração dos aplicativos e interação dos usuários. Assim, os programadores de nível mais alto se tornam mais produtivos, dispondo de ferramentas muito mais poderosas de se trabalhar (3).

Segundo (6), em outros aspectos, um sistema operacional pode ser considerado um alocador de recursos. Considerando que sistemas computacionais apresentam diversos recursos - como memória principal, processador, dispositivos de E/S, etc -, o sistema operacional atua como gerente deles e os aloca a usuários e programas, conforme sua necessidade, a fim de que as tarefas sejam executadas de maneira justa.

Muitos programas e usuários podem requisitar recursos limitados de maneira conflitante. Assim, o sistema operacional é responsável por decidir a quais pedidos e quando serão concedidos os recursos.

Outra visão, por fim, de sistemas operacionais, enfatiza seu papel de controle sobre dispositivos de E/S e programas de usuário, onde ele controla a execução dos programas de usuários para evitar erros, preocupando-se especialmente com a operação e o controle de dispositivos de entrada e saída.

O presente trabalho, portanto, se divide da seguinte maneira: o Capítulo 2 descreve os objetivos gerais e específicos de desenvolvimento do projeto; o Capítulo 3 apresenta

os conceitos teóricos por trás do sistema operacional a ser implementado; o Capítulo 4 apresenta os métodos e técnicas utilizados para o desenvolvimento do sistema, o Capítulo 5 apresenta os testes e resultados obtidos do sistema e o Capítulo 6 traz comentários acerca do projeto desenvolvido e da expectativa para projetos posteriores.

2 Objetivos

2.1 Geral

Elaborar um sistema operacional que execute programas sobre o processador desenvolvido na disciplina *Laboratório de Sistemas Computacionais: Arquitetura e Organização de Computadores* de forma singular e paralela (preempção), que permita a interação com um sistema de arquivos e que permita a interação com o usuário através de uma interface.

2.2 Específico

Apresentar, definir e implementar o projeto de um sistema operacional que gerencie:

- Processos:
  - Escalonamento de CPU (troca de contexto);
- Memória:
  - Alocação;
- Sistema de arquivos: prover métodos essenciais para criar, editar, renomear e eliminar arquivos e diretórios.
- Dispositivos de Entrada e Saída:
  - Controle do Sistema sobre operações de entrada e saída.

3 Fundamentação Teórica

3.1 Processos

Os conceitos não referenciados diretamente apresentados na presente seção são oriundos de (3).

Um processo consiste, segundo (3), em uma abstração de um programa em execução.

Em sistemas multiprogramados, a CPU alterna de programa em programa rapidamente, trazendo uma sensação de multitarefa ao usuário. A fim de prover controle sobre a execução de múltiplas atividades, surgiu o modelo de processo.

Nele, todos os *softwares* executados se organizam em processos, os quais consistem não apenas nos programas em execução, mas também acompanham os valores atuais do contador de programa, registradores e variáveis - algo como uma foto do estado do processador.

Assim, diferentes processos apresentam, cada um, seu fluxo de controle. Apesar de haver apenas um contador de programa físico, cada processo guarda seu próprio contador de programa virtual; quando é executado, o processo carrega seu valor de contador de programa ao processador. Ao terminar seu tempo de CPU, o valor do contador de programa físico atual é carregado de volta ao contador de programa virtual.

Portanto, um único processador pode ser compartilhado entre diversos processos, desde que um algoritmo de escalonamento determine os momentos em que o trabalho deve ser cessado para um processo, e iniciado para outro.

3.1.1 Estados de processos

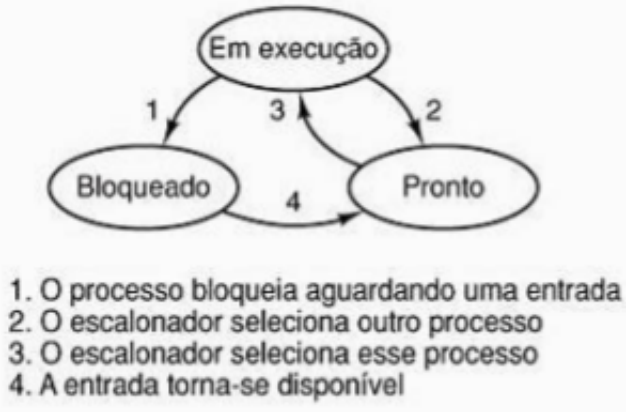
Os conceitos não referenciados diretamente apresentados na presente seção são oriundos de (3).

Processos, por vezes, precisam interagir com outros processos, como quando a entrada de um processo consiste no resultado de saída de outro. A fim de se organizar a interação entre os processos dentro da CPU, atribuem-se estados a eles, os quais podem ser:

- Em execução: utilizando a CPU;
- Pronto: disponível para execução conforme o arbítrio da CPU;
- Bloqueado: aguardando um evento externo para prosseguir.

A Figura 2 ilustra os possíveis estados e como eles transitam entre si.

Figura 2 – Possíveis estados que um processo pode assumir.



Fonte: Modern Operating Systems (3)

O gerenciamento da execução desses processos é feito pelo escalonador, o qual rege todo o tratamento de interrupção e detalhes sobre a inicialização e bloqueio de processos.

3.1.2 Implementação de Processos

Os conceitos não referenciados diretamente apresentados na presente seção são oriundos de (3).

A implementação do modelo de processos é realizada pelo sistema operacional com o auxílio da tabela de processos, a qual apresenta uma entrada para cada processo e contém as informações apresentadas na Tabela 1. Estas informações são necessárias para que, ao passar do estado "em execução"para o estado "pronto"ou "bloqueado", o processo possa retornar à execução de onde parou.

Associada aos dispositivos de entrada e saída, existe uma parte da memória conhecida como **arranjo de interrupções**. Este contém os endereços das rotinas dos serviços de interrupção.

O tratamento de interrupção e o escalonamento acontecem, basicamente, da seguinte forma:

1. O *hardware* adiciona o contador de programa à pilha;
2. O novo contador de programa é carregado do arranjo de interrupções pelo *hardware*;
3. Os registradores são salvos por um procedimento em linguagem de baixo nível;
4. Esse mesmo procedimento configura uma nova pilha;

Tabela 1 – Exemplo de campos comuns a um processo.

Tamanho (bits)
Registros
Contador de programa
Palavra de estado do programa
Ponteiro da pilha
Estado do processo
Prioridade
Parâmetros de escalonamento
ID do processo
Processo pai
Grupo de processo
Sinais
Momento de início do processo
Tempo de CPU
Tempo de CPU do processo filho
Tempo do alarme seguinte

Fonte: Modern Operating Systems (3)

5. O serviço de interrupção em linguagem de nível mais alto - comumente C - lê e armazena a entrada temporariamente;

6. O escalonador elege o próximo processo a ser executado;

7. O procedimento em alto nível retorna para o código em baixo nível;

8. O procedimento em baixo nível inicia o novo processo.

3.2 Comunicação Entre Processos

Os conceitos não referenciados diretamente apresentados na presente seção são oriundos de (3).

Há situações em que processos se comunicam entre si, como quando a entrada de um processo consiste na saída de outro; essa comunicação deve ocorrer sem interrupções e de forma estruturada.

Para que processos se comuniquem apropriadamente, há três fatores a se considerar:

1. Como um processo passa informações para outro;

2. Como garantir que múltiplos processos não entrem em conflito por recursos;

3. Como garantir que os processos executem em uma ordem coerente com as dependências que possuem entre si.

3.2.1 Condições de Corrida e Regiões Críticas

Os conceitos não referenciados diretamente apresentados na presente seção são oriundos de (3).

Há situações em que processos que trabalham juntos podem compartilhar um armazenamento em comum. Em casos como esse, segundo (7), condições de corrida ocorrem quando o resultado de um cálculo pode ser afetado por uma sequência de eventos, de maneira que o arquivo compartilhado é alterado por um processo diferente de maneira indesejável.

Para que se evitem situações como essa, é necessário que haja uma exclusão mútua, onde o recurso deve ser utilizado por um processo de cada vez. As seções do código que incorporam recursos que devam ser utilizados por um processo de cada vez são conhecidas como regiões críticas, e para que sejam implementadas com sucesso, algumas condições devem ser satisfeitas:

1. Dois processos nunca podem estar na região crítica simultaneamente;

2. Um processo fora de sua região crítica nunca pode bloquear outros processos.

3. Um processo não pode esperar eternamente para entrar em sua região crítica.

3.3 Escalonamento

Os conceitos não referenciados diretamente apresentados na presente seção são oriundos de (3).

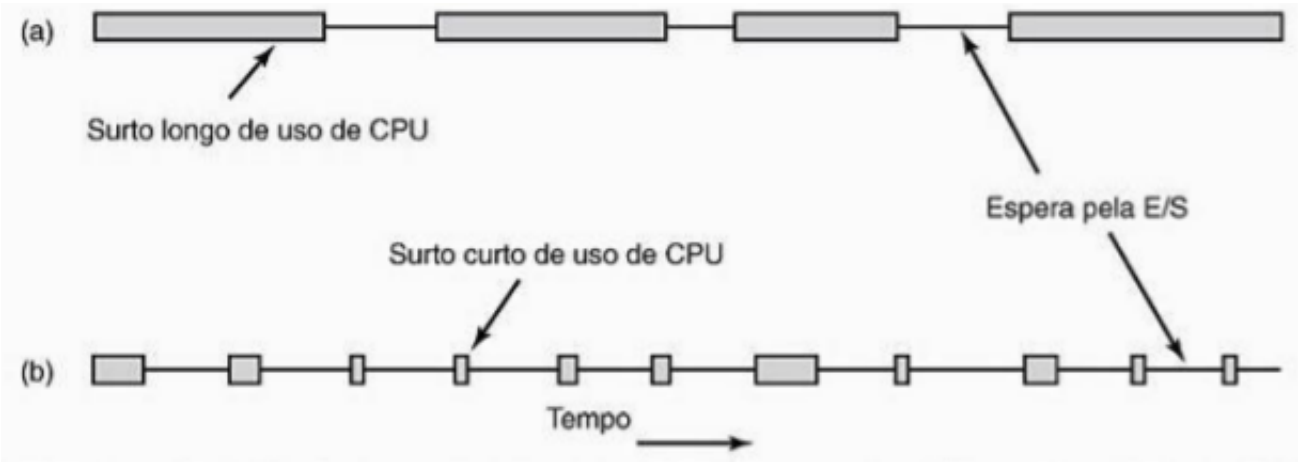
Computadores multiprogramados, por vezes, apresentam múltiplos processos competindo pela CPU, de modo que dois ou mais processos podem estar no estado "pronto"ao mesmo tempo. O sistema operacional, portanto, precisa escolher qual processo será executado; a parte dele que faz isso é o escalonador, o qual apresenta um algoritmo próprio de escalonamento.

Além de eleger o processo a ser executado, o escalonador também se preocupa em utilizar a CPU eficientemente, uma vez que o processo de chaveamento é relativamente custoso.

Nele, a princípio, ocorre o chaveamento do modo usuário para o modo *kernel*. Sendo salvo o estado atual do processo, então, um novo processo é selecionado pela execução do algoritmo de escalonamento. A unidade de gerenciamento da memória - *memory management unit* -, em seguida, precisa ser carregada com o mapa de memória do novo processo. Por fim, então, o novo processo é iniciado.

Processos, geralmente, alternam entre piques de computação e requisições de entrada e saída, como se observa na Figura 3. Na letra (a), é possível perceber que certos processos gastam mais tempo computando que outros - sendo esses, os "limitados pela CPU-, enquanto outros passam mais tempo esperando entrada e saída (b) - os "limitados pela E/S".

Figura 3 – (a) Processo limitado pela CPU; (b) processo limitado pela E/S.



Fonte: Modern Operating Systems (3)

O escalonamento pode ser necessário em uma miríade de situações. Entre elas, temos:

- Ao se criar um novo processo, é necessário escolher se o processo pai ou o processo filho será executado, uma vez que ambos estão "prontos";
- Quando um processo é terminado, onde seu lugar na CPU deve ser cedido a outro processo que esteja "pronto";
- Quando um processo é bloqueado, outro deve ser elegido para executar;
- Quando um processo retorna de uma operação de E/S e se encontra "pronto".

Um algoritmo de escalonamento com preempção deve escolher um processo a ser executado por um tempo máximo. Caso ele continue a executar após esse intervalo de tempo, o escalonador elegerá outro processo para ser executado.

Algoritmos de escalonamento podem ser: em lote, interativos ou em tempo real.

Ao se considerar a natureza do projeto desenvolvido, a ele se aplicam algoritmos de escalonamento interativos, uma vez que fornecem:

- Justiça: uma porção justa da CPU para cada processo;

- Equilíbrio: todas as partes do sistema se mantêm ocupadas;

- Tempo de resposta: as requisições são respondidas rapidamente;

- Proporcionalidade: as expectativas dos usuários são satisfeitas.

3.4 Memória

Os conceitos não referenciados diretamente apresentados na presente seção são oriundos de (3).

Um sistema operacional é dotado de um gerenciador de memória; este gerencia a memória eficientemente, mantendo controle de quais partes da memória estão ou não sendo utilizadas, alocando memória aos processos quando necessário e liberando após o término deles.

A maior parte dos sistemas operacionais modernos trabalha com uma memória virtual, onde cada programa tem seu próprio espaço de endereçamento, que se divide em páginas. Páginas consistem em séries contíguas de endereços e são mapeadas na memória física.

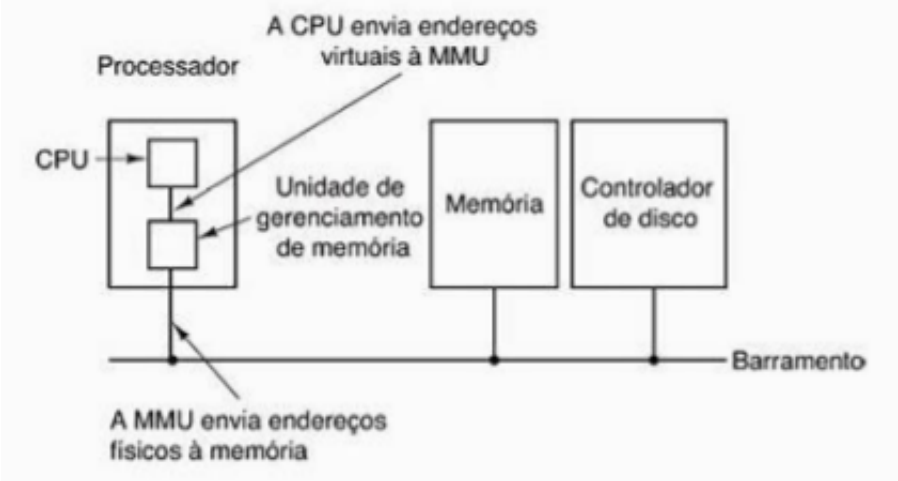
Quando uma parte do espaço de endereçamento de um programa que está n memória é referenciada, o *hardware* executa dinamicamente o mapeamento necessário.

3.4.1 Paginação

Os conceitos não referenciados diretamente apresentados na presente seção são oriundos de (3).

Programas, ao serem executados, podem gerar um conjunto de endereços de memória. Eles são denominados endereços virtuais, constituindo o espaço de endereçamento virtual. Quando a memória virtual é acionada, o endereço virtual é passado a uma unidade de gerenciamento de memória (MMU), a qual mapeia endereços virtuais a endereços físicos, como se observa na Figura 4

Figura 4 – Posição e função da unidade de gerenciamento de memória.



Fonte: Modern Operating Systems (3)

Na paginação, o endereçamento virtual se divide em unidades que se denominam páginas, enquanto as molduras correspondentes na memória física se chamam molduras de página.

O mapeamento dos endereços virtuais em físicos ocorre de forma que o endereço virtual se divide em um número de página virtual e um deslocamento. Nesse caso, o número da página virtual é um índice dentro da tabela de páginas que permite que se encontre a entrada da tabela associada à pagina virtual requerida.

O número da moldura de página física é encontrado a partir dessa entrada, e então é concatenado aos *bits* do deslocamento, a fim de formar o endereço físico a ser enviado à memória.

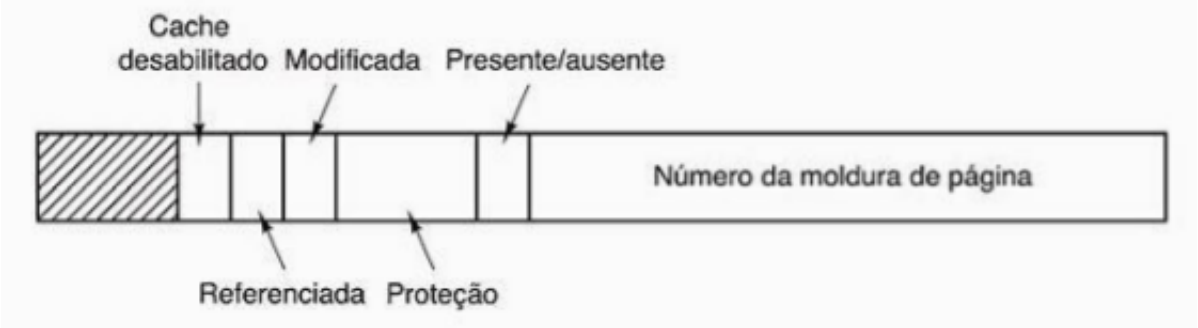
A estrutura que guarda tais correspondências é denominada "tabela de páginas", e mapeia as páginas virtuais em molduras de página físicas. A Figura 5 apresenta um exemplo de entrada de tabela de páginas. Nela, o campo de destaque é o "número da moldura de página", uma vez que o objetivo do mapeamento de páginas é localizar esse valor. O *bit* "presente/ausente"indica, caso seja 1, que a entrada será válida e poderá ser usada; se ele for 0, indicará a ausência da página virtual na memória principal naquele instante.

Os *bits* "modificada"e "referenciada"indicam o estado da página, onde "modificada"indica se ela foi alterada e precisa de atualização no disco, e "referenciada"indica quando a página física é referenciada para leitura e escrita, ajudando o sistema operacional na escolha de uma página a ser substituída quando há falta de páginas, uma vez que as páginas físicas que não estão sendo utilizadas são as melhores candidatas para substituição.

O último *bit*, em sistemas que suportam esta funcionalidade, permite que o meca-

nismo de *cache* seja desabilitado para a página mencionada.

Figura 5 – Entrada de uma tabela de páginas.



Fonte: Modern Operating Systems (3)

3.5 Sistema de Arquivos

Os conceitos não referenciados diretamente apresentados na presente seção são oriundos de (3).

Um arquivo, segundo (3), é um mecanismo de abstração que oferece meios de armazenar em e ler informações do disco. Ao criar um arquivo, um processo lhe atribui um nome; ao terminar o processo, o arquivo continua a existir, e pode ser acessado por outros processos através de seu nome. Os arquivos comumente apresentam extensões, as quais evidenciam a qual tipo pertencem. As extensões são escritas após um ponto; como, por exemplo, em "*arquivo.txt*", onde "arquivo"é o nome do arquivo e ".txt"é sua extensão, a qual convencionalmente indica um arquivo de texto.

Uma forma bastante convencional de estruturação de sistema de arquivos é aquele em que o significado do arquivo é atribuído em nível de usuário; tal estratégia é adotada tanto por sistemas *UNIX* quanto pelo *Windows*. Nesse caso, o sistema operacional trata do arquivo como apenas uma cadeia de *bytes* e, portanto, a flexibilidade de criação e uso de arquivos é maior.

Sistemas operacionais costumam dar suporte a diversos tipos de arquivos. Os tipos que se vale destacar são:

- Arquivos regulares: são aqueles que contêm informação do usuário;
- Diretórios: arquivos do sistema que guardam a estrutura do sistema de arquivos;
- Arquivos especiais de caracteres: são inerentes à entrada e saída, sendo utilizados a fim de modelar dispositivos de E/S;
- Arquivos especiais de blocos: são utilizados na modelagem de discos.

3.5.1 Operações com Arquivos

Os conceitos não referenciados diretamente apresentados na presente seção são oriundos de (3).

Cada sistema pode oferecer diferentes operações para que se armazenem e se recuperem informações por meio de arquivos. Abaixo se descrevem operações comuns e que hão de ser implementadas no sistema descrito neste relatório:

- Criar: cria-se o arquivo e definem-se seus atributos, porém sem dados envolvidos;
- Eliminar: uma chamada de sistema remove o arquivo indesejado para liberar espaço em disco;
- Abrir: a fim de que um arquivo seja usado, um processo o abre para que o sistema busque e coloque na memória principal os atributos e endereços inerentes ao arquivo;
- Fechar: quando não é mais necessário utilizar um arquivo, ele deve ser fechado, de maneira que seus atributos e endereços liberem espaço na memória principal;
- Ler: é realizada a leitura dos dados no arquivo;
- Escrever: escrevem-se dados no arquivo;
- Obter nome: o nome do arquivo é obtido;
- Renomear: altera-se o nome do arquivo.
- Mover: move-se o conteúdo de um arquivo para outro diretório;
- Copiar: o conteúdo do arquivo é copiado para outro arquivo.

3.5.2 Diretórios

Os conceitos não referenciados diretamente apresentados na presente seção são oriundos de (3).

O controle sobre a organização de arquivos se dá por meio de diretórios. Os sistemas de arquivos modernos apresentam diretórios hierárquicos, os quais permitem que arquivos relacionados sejam agrupados em um mesmo local. Assim, uma árvore de diretórios se forma conforme o usuário cria diretórios dentro de diretórios nos mais diversos níveis.

Quando se organiza um sistema de arquivos dessa forma, é necessário que se especifique o nome dos arquivos. Uma forma bastante usual é a em que se é dado um nome de caminho absoluto para cada diretório, onde o arquivo é encontrado quando se referencia o caminho que leva a ele. Um exemplo de caminho é o dos sistemas UNIX, onde uma barra indica a separação entre os diretórios e o último nome indica o nome do arquivo, da

seguinte forma: "/usr/ast/arquivo.txt". Tais caminhos sempre partem do diretório raiz do sistema.

Tal qual arquivos, diretórios podem ser criados, excluídos, abertos, fechados, lidos e renomeados.

3.6 Dispositivos de Entrada e Saída

Os conceitos não referenciados diretamente apresentados na presente seção são oriundos de (3).

Um sistema operacional deve controlar os dispositivos de entrada e saída que interagem com um sistema computacional, emitindo comandos, interceptando interrupções, e fornecendo uma interface simples e amigável entre os dispositivos e o restante do sistema.

Os dispositivos de E/S se dividem em duas categorias: dispositivos de blocos, os quais armazenam informação em blocos; dispositivos de caracteres, os quais enviam ou recebem fluxos de caracteres, sem levar em consideração estruturas de blocos. Como exemplos do primeiro, temos CD-ROM's e *pendrives*; como exemplos do segundo, temos impressoras e *mice*.

As unidades de entrada e saída costumam apresentar dois componentes distintos, sendo um mecânico e o outro, eletrônico. O componente eletrônico é chamado de "controlador do dispositivo". Seu trabalho é converter o fluxo de *bits*, o qual é serial, em blocos, e verificar os dados por erros.

Controladores apresentam registradores para se comunicarem com a CPU, de forma que o sistema operacional pode interagir com o dispositivo,

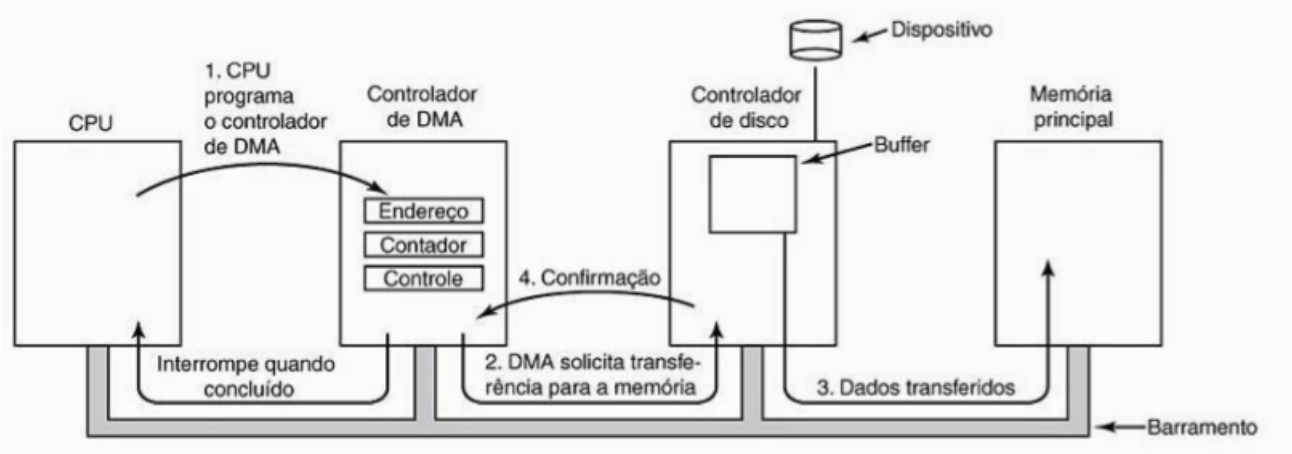
3.6.1 Acesso Direto à Memória (DMA)

Os conceitos não referenciados diretamente apresentados na presente seção são oriundos de (3).

A fim de trocar dados com dispositivos de entrada e saída, a CPU precisa endereçar os controladores dos dispositivos. A fim de que isso seja feito de maneria direta e eficiente, existe o "acesso direto à memória"(DMA).

O controlador de DMA pode acessar diretamente o barramento do sistema independente da CPU; ele contém vários registradores acessíveis à CPU, incluindo um de endereçamento de memória, um contador de *bytes* (para verificação de erros) e ainda um ou mais registradores de controle. Estes especificam qual porta de entrada ou saída está em uso, a direção, unidade e número de *bytes* na transferência. A Figura 6 ilustra um exemplo de operação de transferência realizada utilizando-se um DMA.

Figura 6 – Operação de transferência com DMA.



Fonte: Modern Operating Systems (3)

4 Desenvolvimento

4.1 Preparação e Uso de Ferramentas

O projeto de um sistema operacional é complexo, de maneira que os componentes devem funcionar corretamente e de maneira conjunta. Portanto, houve um cuidado especial dentro do desenvolvimento deste projeto sobre a organização e estruturação do trabalho.

O desenvolvimento foi realizado dentro de um modelo ágil, onde um planejamento geral de alto nível foi realizado antes do início das atividades, mas o planejamento específico e de nível mais baixo foi realizado dentro das etapas, conforme suas necessidades. Assim também, as etapas foram divididas em *milestones*, onde cada uma delas representa um entregável de valor e incremental; ou seja: o material entregado em uma *milestone* tanto tem valor por si só quanto é útil para o desenvolvimento da etapa seguinte.

As *milestones* foram divididas em iterações e as iterações foram divididas em atividades. Há como se observar com mais detalhes esta metodologia dentro do *Kanban* criado para este projeto (8).

A explicação do desenvolvimento do projeto será feita de acordo com a forma que ele foi desenvolvido. Portanto, as seções a seguir acompanham as *milestones*:

- Escopo e Estrutura 4.2;
- Milestone #0: Executar um Programa 4.3
- Milestone #1: Trocar Contexto Manualmente 4.4;
- Milestone #2: Trocar Contexto Automaticamente 4.5;
- Milestone #3: Operações com arquivos 4.6.

Para detalhes técnicos, o projeto possui um repositório no *git* (9), assim como o processador sobre o qual ele foi desenvolvido (1) e o *assembler* (2) criado para a codificação. As alterações feitas no processador mencionado se descrevem conforme sua realização dentro de suas respectivas *milestones*.

4.2 Escopo e Estrutura

Esta etapa se refere a como o processador (1) foi inicialmente preparado para a execução do sistema operacional, assim como a forma como seus recursos foram divididos e abstraídos para o uso deste.

4.2.1 Implementação do Disco Rígido

O disco rígido foi implementado como uma memória de vetor simples. A abstração de trilha e setor foi feita para o usuário através do *assembler*, onde o usuário tem condições de informar a trilha e setor de maneira separada e este a resolve para o processador com instruções já existentes. Sua estrutura de implementação é detalhada na Subseção 4.2.3.

4.2.2 BIOS

A BIOS foi escrita como uma memória simples e realiza os seguintes passos:

1. Teste de *output*;
2. Teste de memória;
3. Transferência do Sistema Operacional para a memória de instruções;
4. Início do Sistema.

Para a transferência do sistema operacional do disco rígido para a memória de instruções, foi criada uma instrução específica que escreve o conteúdo de um registrador do banco de registradores na memória de instruções. Assim, um laço dentro da BIOS percorre o disco rígido - desde o início, onde fica o sistema operacional - até a condição de parada de instrução nula. Esse laço transfere instrução a instrução do disco rígido para o banco de registradores, e deste para a memória de instruções.

Dois componentes e uma instrução foram criados para mediar a transferência do controle sobre a execução da BIOS para a memória de instruções: o *instructions source selector*, o *instructions multiplexer* e a *start system*, respectivamente. Desta forma, a instrução *start system* nada mais faz do que alterar o estado do *instructions source selector* de 0 para 1 - mudando a fonte de instruções do sistema da BIOS para a memória de instruções - e reiniciando contador de programa para que o sistema operacional possa rodar.

4.2.3 Implementação das Memórias

A forma como as memórias do sistema haviam sido implementadas no projeto original do processador (1) trazia uma grande desvantagem no tempo de compilação do sistema e até mesmo na capacidade deste. Por isso, as memórias foram modificadas para padrões que o *Software Altera Quartus II* pudesse inferir e, em sua compilação, aproveitar de forma mais eficiente os recursos do *kit FPGA*.

A memória de instruções e a memória de dados foram alteradas de modo que sua escrita ocorre no final de um *clock* de escrita, o qual foi alterado para ser mais lento, e

sua leitura ocorre no fim de um *clock* específico de leitura. Essa implementação também é o que permite a instrução *store_i_ram*, a qual lê o conteúdo de um registrador e o guarda em uma posição dentro da memória de instruções. A seguir, a estrutura desta implementação em *verilog*:

```
module modulo_de_memoria(clock_de_leitura, clock_de_escrita, endereco_de_leitura,
    endereco_de_escrita, saida, entrada, escrever_na_memoria);
         input [31:0] entrada;
         input [11:0] endereco_de_leitura;
         input [11:0] endereco_de_escrita;
         input clock_de_escrita;
         input clock_de_leitura;
         input escrever_na_memoria;
         output reg [31:0] saida;


         reg [31:0] memoria[1700:0];


         always @ ( negedge clock_de_escrita ) begin
//escrita
                        if (escrever_na_memoria) begin
                                memoria[endereco_de_escrita] <= entrada;
                        end
         end

                 always @ ( posedge clock_de_leitura) begin
//leitura
                                saida <= memoria[endereco_de_leitura];
         end

endmodule
```

O disco rígido, porém, utiliza apenas um valor de *clock* - no caso, utilizou-se o mesmo do *clock* de leitura dos outros componentes. Este, também, possui uma seção com valores pré-escritos, determinados através de um *initial*:

```
module disco_rigido(entrada, endereco, clock, saida, flag_escrever_hd);
  input  [31:0] entrada;
  input [20:0] endereco;
  input flag_escrever_hd;
  input clock;
  output [31:0] saida;

        reg [31:0] HD[12000:0];
  reg [20:0] endereco_local;


        //conteudo pre-escrito
        initial begin
    HD[0] = 32'b01010100000000000000000100000001;//exemplo de instrucao

    end


    always @ (posedge clock)
    begin
      // escrita
```

```
22        if (flag_escrever_hd)
23          HD[endereco] <= entrada;
24
25        endereco_local <= endereco;
26      end
27
28          assign saida = HD[endereco_local];
29
30  endmodule
```

4.2.4 Programas e Processos

Programas são arquivos dentro do sistema que apresentam identificação e conteúdo. No caso do sistema presente, eles guardam as seguintes propriedades:

- ID - um número que deve ser maior que 10 e também representa seu nome;
- Índice de Programa - o índice de onde se encontra seu conteúdo na lista de programas do disco rígido 4.2.5;
- Conteúdo - instruções que compõem o programa.

Um **processo** é um programa em execução. Neste sistema, processos possuem os seguintes atributos:

- ID - o mesmo que possui quando é um programa;
- Índice de Processo - corresponde à posição de seu conteúdo na memória de dados 4.2.6;
- Estado - como se encontra para o Sistema Operacional.

Um processo pode assumir diferentes valores de acordo com a Tabela 2:

Um conceito importante do sistema é o de **processo protagonista**. Este é o processo selecionado pelo usuário e que se apresenta em primeiro plano para ele.

Um processo faz uso da memória de instruções, onde executa, da memória de dados, onde guarda suas variáveis, e do banco de registradores. A fim de que ocorram a execução e a troca de contexto, o sistema operacional precisa guardar o estado do processo antes de parar de executá-lo. Assim, as seguintes informações acerca dele devem ser guardadas:

- Estado do banco de registradores - Lista de informações de programas e processos 4.2.5;
- Estado do Contador de Programa - Lista de informações de programas e processos 4.2.5;

Tabela 2 – Estados de um Processo

Estado	Valor	Comentário
Inexistente	0	Não há processo para o programa
Executando ou Pronto	1	Se estiver no início da lista de processos, estará executando; se não, pronto
Bloqueado	2	Esperando por uma entrada ou saída do usuário

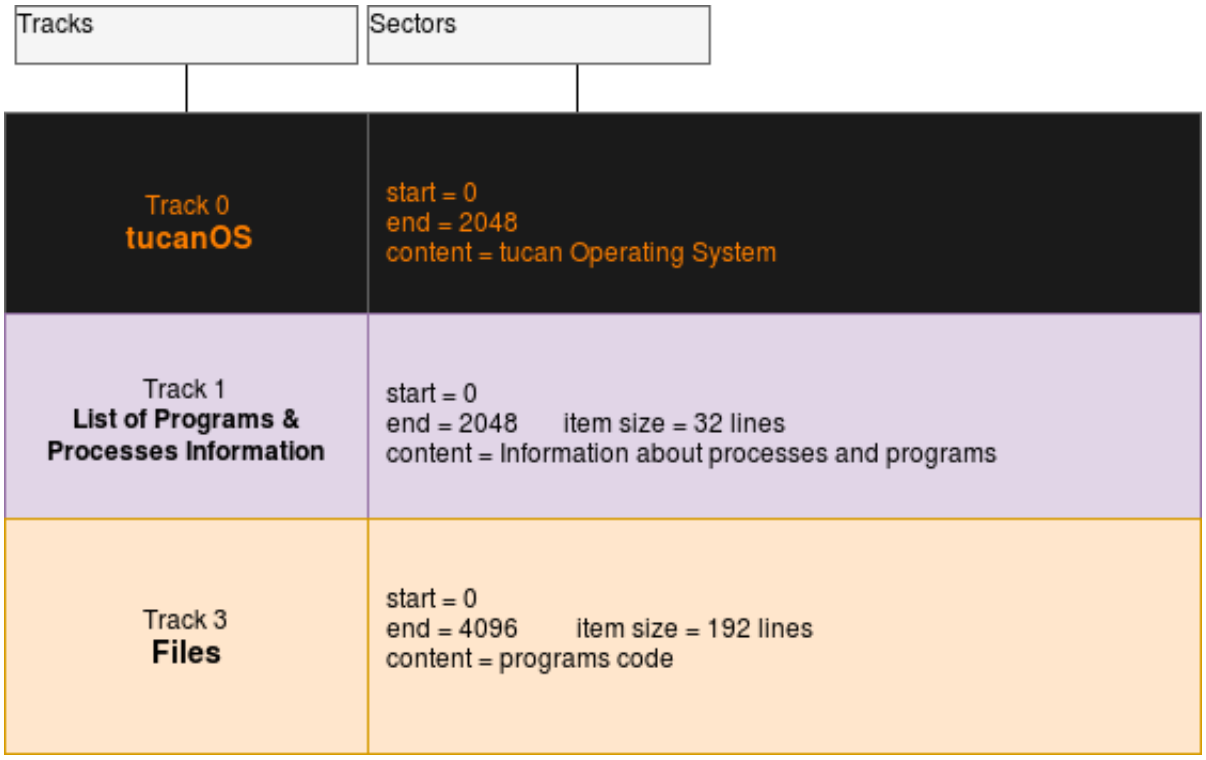
Fonte: O Autor

- Variáveis de Programa - Lista de dados de programas 4.2.6.

4.2.5 O Disco Rígido para o Sistema Operacional

O disco rígido se divide conforme a Figura 7.

Figura 7 – Estrutura do Disco Rígido.



Fonte: O Autor

A **Trilha 1** guarda o Sistema Operacional.

A **Trilha 2** guarda a Lista de Informações de Programas e Processos, onde ficam juntos os atributos de programas e processos, assim como dados específicos do processo:

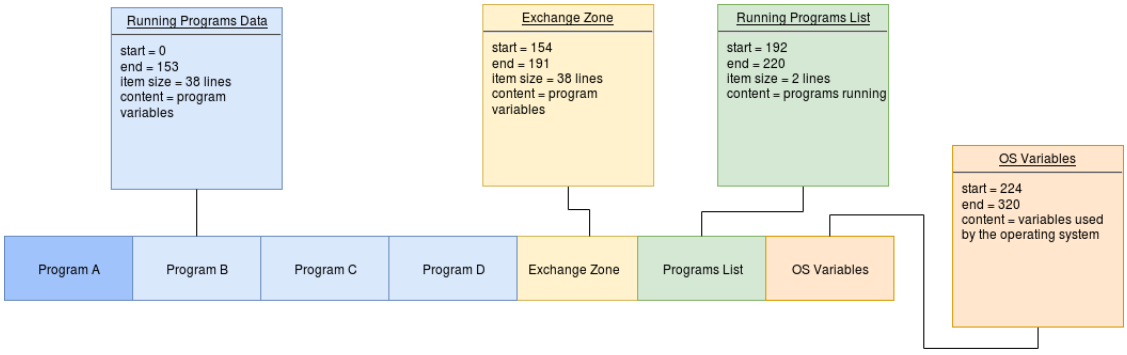
- ID;
- Índice de Processo;
- Estado;
- Índice de Programa;
- Contador de Programa;
- Conteúdo do Banco de Registradores.

A **Trilha 3** guarda as instruções de cada programa.

4.2.6 As Memórias para o Sistema Operacional

A **memória de dados** se divide conforme a Figura 8.

Figura 8 – Estrutura da Memória de Dados.



Fonte: O Autor

A **memória de instruções** se divide em apenas dois setores:

- Programa em Execução - da linha 0 à linha 255;
- Sistema Operacional - da linha 256 à linha 1600.

4.2.7 O Banco de Registradores para o Sistema Operacional

Nesta implementação, certos registradores foram reservados para o uso do sistema operacional, enquanto outros foram reservados para o uso do programa em execução. Essa organização foi feita para que o sistema operacional tenha autonomia para trabalhar sem alterar o estado do programa em execução e vice e versa.

Para o sistema operacional, o banco de registradores está organizado da seguinte maneira:

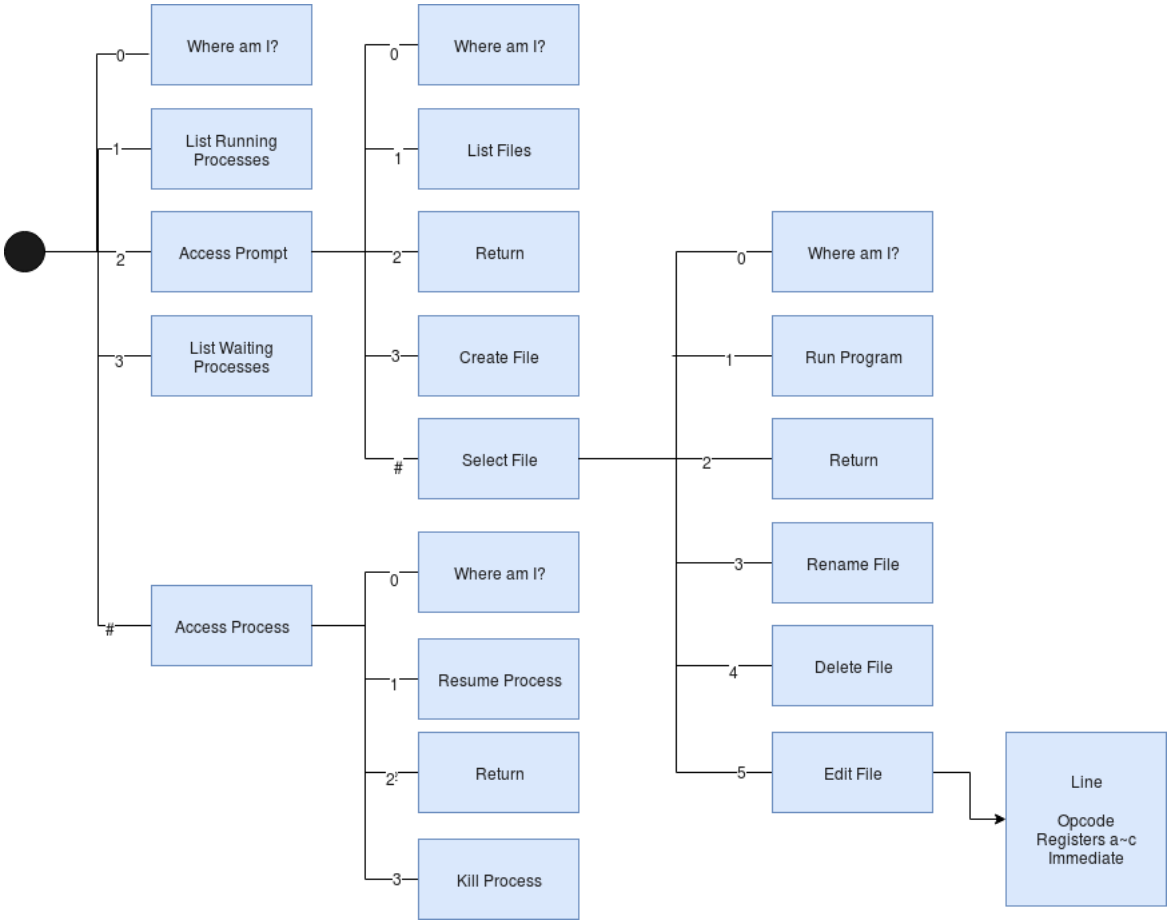
- Registrador 0: sempre de valor zero para possíveis comparações;
- Registradores 1 a 19: permitidos para o uso dos programas em execução;
- Registradores 20 a 32: reservados para uso do Sistema Operacional.

Vale a pena ressaltar que o registrador de número 28 foi utilizado para guardar continuamente o contador de programa do programa em execução, o que será abordado com mais detalhes na Seção 4.4.

4.2.8 A *Prompt* de Comando

O sistema foi inteiramente desenvolvido em torno da forma como haveria de ser utilizado pelo usuário. Assim, a Figura 9 apresenta os comandos que o usuário deve inserir para realizar as operações que desejar.

Figura 9 – Mapa da Prompt de Comando.



Fonte: O Autor

4.3 *Milestone* #0: Executar um Programa

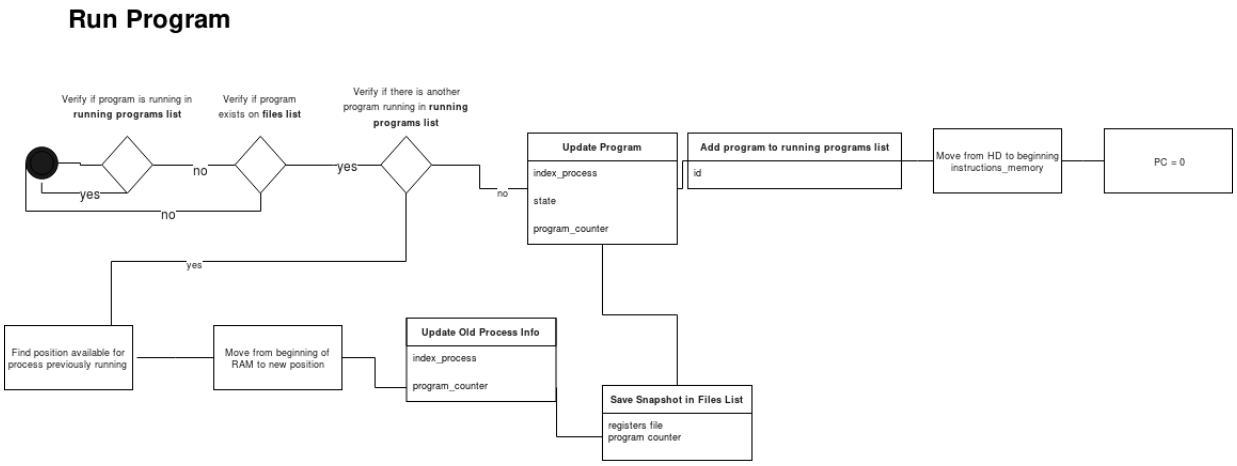
A execução de um programa pode se realizar em dois casos específicos:

1. Ele é o primeiro programa a ser executado;

2. Já há outro(s) programa(s) executando.

Esses dois casos foram divididos em iterações para tornar a implementação incremental. A Figura 10 apresenta todos os passos para que um programa seja executado.

Figura 10 – Fluxo para execução de programas.



Fonte: O Autor

Assim, para que um programa seja executado, ele precisa estar parado e existir na lista de arquivos. Verifica-se, então, se há outros programas rodando.

- Se sim, Iteração 1: atualiza informações do processo novo (índice de processo <= 0, estado <= 1, contador de programa <= 0), adiciona o processo à lista de processos, move as instruções do disco rígido para a memória de instruções e salta o contador de programa para 0 (início das instruções do programa).

- Se não, Iteração 2: guarda dados do programa rodando na próxima posição de memória disponível, guarda estado do banco de registradores e contador de programas do programa rodando em sua posição da Lista de Informações de Programas e Processos 4.2.5, e segue de maneira análoga à iteração 1.

Nota-se que se deve guardar o programa que foi selecionado para execução como o protagonista.

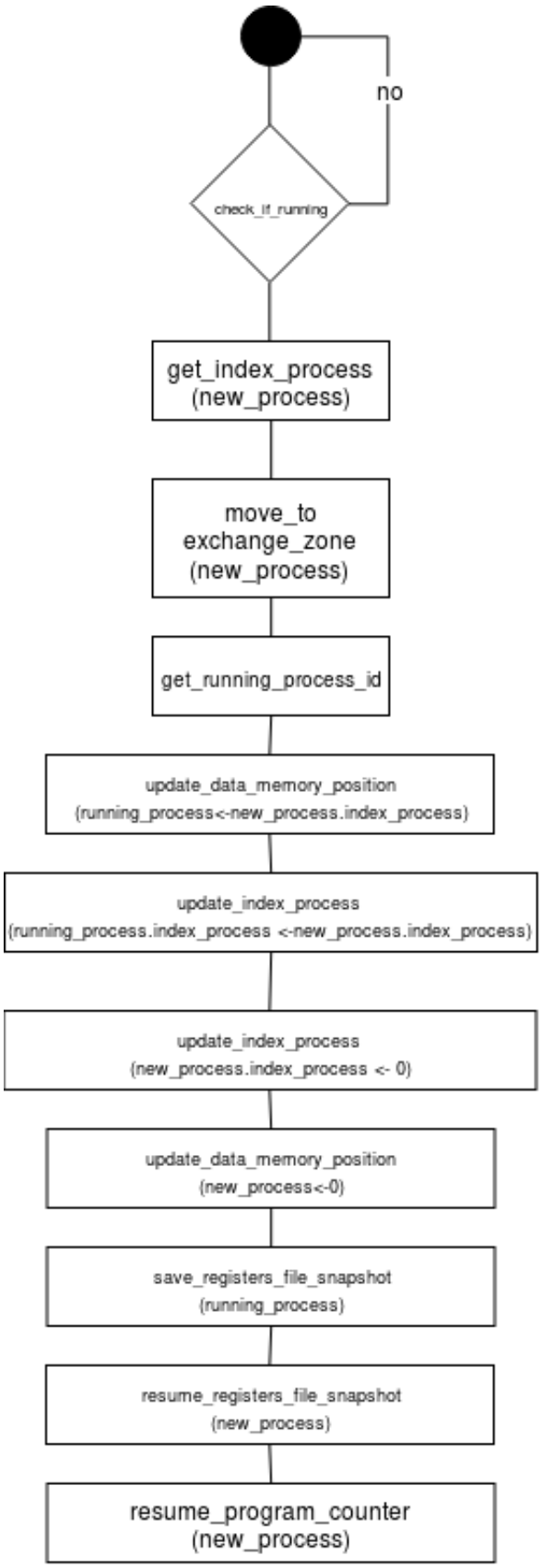
4.4 *Milestone* #1: Trocar Contexto Manualmente

Na troca de contexto manual, durante a execução do programa, o usuário deve ser capaz de voltar para a *prompt* de comando e trocar para outro programa que já está em execução.

Este processo guarda semelhanças com a execução quando um programa já está rodando, portanto aproveita muitas das funções presentes na *milestone* anterior.

A Figura 11 apresenta os passos do sistema na troca de contexto manual.

Figura 11 – Fluxo para troca de contexto manual.



Fonte: O Autor

Ela ocorre de maneira que os dados do programa que se deseja resumir são primeiro guardados na zona de troca da memória de dados 4.2.6. Os dados do programa que está em execução são, então, transferidos para a posição da memória de dados em que o programa a ser resumido estava. Os dados do programa a ser resumido são, então, transferidos para a posição 0 da memória de dados (execução), o estado do processador para o programa anterior é guardado e o estado para o programa resumido é resumido.

A abordagem para guardar o contador de programa foi a seguinte: ele é continuamente atribuído a um registrador reservado só para isso, e a atribuição ocorre apenas quando um programa está executando - se é um PC acima de 256, ele não atualiza, pois é o sistema operacional que está executando. Assim, quando o usuário (ou o sistema) voltam para o sistema operacional, ele continua inalterado para ser guardado nas informações do programa.

Há valores que devem ser guardados de maneira diferenciada para não gerar problemas no fluxo de dados do sistema. Portanto, o contador de programa passa por um *filtro* antes de enviado ao banco de registradores.

Se trata de um módulo do processador (1) que faz uma conversão simples:

- Para instruções *jump*, *jumpr* e *pbranch*: contador de programa <= contador de programa;
- Para instruções *branchz*, *branchn*: contador de programa <= contador de programa - 1;
- Para o restante das instruções: contador de programa <= contador de programa + 1.

O programa selecionado manualmente para ser resumido se torna o programa protagonista 4.2.4.

4.5 *Milestone* #2: Trocar Contexto Automaticamente

O fluxo da troca de contexto automática é o mesmo da manual, porém ela acontece sem que o usuário escolha ou perceba, nunca invadindo a noção do usuário do programa protagonista 4.2.4. Assim, a entrada e saída e a finalização de programas devem ser controladas, assim como os programas que estarão executando a cada período.

Para dar esse controle ao sistema operacional, se desenvolveu um módulo que o alimenta com informações do sistema de tempos em tempos. Seu nome é *tucanos watchdog*, ou *cão de guarda do tucanos*. Ele é uma máquina de estados que apresenta três estados distintos:

1. Contando;
2. Pré-entrada/saída;
3. Halt.

4.5.1 Estado Contando

Seu estado inicial é o *contando*. A cada ciclo de *clock*, ele incrementa até chegar ao valor do *quantum*, aqui definido como 7. Esse incremento só acontece quando um programa está executando; nunca durante a execução do Sistema Operacional.

Ao alcançar o valor, levanta a *flag jump context exchange*. Esta faz com que o contador de programa do sistema vá para uma região específica do sistema operacional que foi desenvolvida para tratar a troca de contexto automática.

Além disso, ele atribui a um registrador interno o valor do suposto próximo índice de processo, onde se o seu valor anterior era 1, ele atribui 2. Se 2, ele atribui 3. Se 3, ele atribui 1. Ele sempre altera esse valor e é o sistema operacional quem vai decidir o que fazer com ele.

Ao entrar em sua seção de troca de contexto automática, o sistema operacional verifica se o valor do registrador interno do *tucanos watchdog* é 1, 2 ou 3.

Caso a posição de memória atribuída esteja vazia, ou o programa ali esteja bloqueado 4.5.2, ele simplesmente volta à execução do programa que já estava rodando. Caso haja um programa *pronto* naquela posição, ele identifica o *id* desse programa e troca o contexto para ele.

4.5.2 Estado Pré-Entrada/Saída

Foi criada uma nova instrução no processador (1) chamada *preio*. No código dos programas, ela deve ser sempre atribuída antes de instruções de entrada e saída, e é o indicador para o sistema de que uma operação de entrada ou saída vai ocorrer.

Quando ela aparece ao longo da execução de um programa, a *flag jump context exchange* é acionada, e o sistema vai para a zona do sistema operacional que lida com troca de contexto.

O valor do **Estado Pré-Entrada/Saída** no registrador interno é *4*, o qual é lido pelo sistema operacional e ativa a rotina que lida com operações de entrada e saída.

Dois casos são possíveis:

1. A instrução apareceu durante a execução do programa protagonista 4.2.4.
2. A instrução apareceu durante a execução de programa simples.

No **Caso 1**, o sistema simplesmente prossegue com a execução do programa protagonista, como se nada tivesse ocorrido.

No **Caso 2**, o estado do programa passa de *em execução* para *bloqueado* e o sistema então troca de contexto de volta ao programa protagonista.

4.5.3 Estado Halt

Quando um programa que está executando alcança uma instrução de *halt*, a troca de contexto também é chamada.

O estado é indicado no registrador interno do *tucanos watchdog* pelo valor *5*.

A rotina referente a esse estado leva o sistema operacional a matar o processo finalizado. Assim, ele altera seu *status* para *parado* e o retira da lista de programas que estão rodando.

Se ele for o protagonista, o sistema retorna ao início da *prompt* de comando. Se não, o sistema troca de contexto de volta ao protagonista.

4.5.4 Problemas de Sincronia

Apesar da lógica correta, o sistema apresentava inconsistências de execução com a troca de contexto automática ligada. Ao se acrescentar, porém, uma mera instrução de saída á rotina de troca de contexto automática, o sistema funcionava perfeitamente.

A correção dessa falha foi feita através de um estado de espera no contador de programa. Assim que a troca de contexto automática é acionada nele, um contador aguarda cerca de meio segundo até que ele inicie de fato as rotinas de troca de contexto automática.

Acredita-se que esse problema seja oriundo de gargalos dentro do fluxo de execução das instruções no sistema operacional em conflito por causa do *clock* muito veloz.

4.6 *Milestone* #3: Operações com Arquivos

Essa *milestone* é alheia às anteriores. Nela, foram implementadas as operações de renomeação, deleção, criação e edição de arquivos.

Para **renomear** um arquivo, o sistema o procura na Lista de Informações de Programas e Processos 4.2.5 segundo seu ID e troca esse valor para o inserido pelo usuário.

A deleção ocorre de maneria semelhante, porém o arquivo é 'renomeado' para *1*, valor que indica uma posição vazia no sistema.

Na criação de um programa, o sistema busca a primeira posição vazia (1) ou a última posição da lista e acrescenta o ID informado àquela posição. Ele também atribui

um valor de índice de programa, caso este estiver nulo.

Na edição, o sistema procura o índice de programa na Lista de Informações de Programas e Processos e sobrescreve a linha escolhida do programa pelo usuário com os valores informados por ele, que são:

1. Opcode [6 bits];
2. Registrador A [5 bits];
3. Registrador B [5 bits];
4. Registrador C [5 bits];
5. Imediato [12 bits].

5 Resultados Obtidos e Discussão

A fim de verificar o funcionamento adequado do sistema, uma série de testes foi escrita e realizada. Acompanhando as funcionalidades do sistema, os testes foram divididos em três fases:

- Testes individuais 5.1 - cada um dos programas de teste foi testado sozinho no sistema ;
- Testes de multitarefa 5.2 - programas foram colocados para executar em paralelo;
- Testes de operações com arquivos 5.3 - apagar, renomear, criar e editar arquivo.

5.1 Testes Individuais

A Tabela 3 apresenta os programas, seus testes, resultados esperados e resultados obtidos. A codificação para os testes se encontram na seção de Anexos 6.

Todos os testes obtiveram os resultados esperados, comprovando que o Sistema Operacional em conjunto ao Processador (1) tem capacidade de executar programas individualmente

5.2 Testes de Multitarefa

O teste de multitarefa foi realizado de maneira que três programas foram executados ao mesmo tempo, segundo os seguintes passos:

1. Um programa foi posto para executar;
2. O usuário alternou para a *prompt* de comando antes que o primeiro programa finalizasse sua execução e executou um segundo programa;
3. Analogamente, o usuário alternou para um terceiro programa e obteve seu resultado;
4. O segundo programa foi selecionado novamente, sua execução foi finalizada e seu resultado foi obtido;
5. O primeiro programa foi selecionado novamente, sua execução terminou e seu resultado foi obtido.

Tabela 3 – Testes individuais sobre Sistema Operacional.

Programa	**Descrição**	**Testes**	**Resultado Esperado**	**Resultado Obtido**	**Resultado Final**
Fibonacci	Dada uma entrada, retorna o valor de sua posição correspondente na sequência de Fibonacci	8	21	21	Sucesso
Maior	Dadas três entradas, retornar qual das duas é a maior	3, 15, 21.	21	21	Sucesso
Divisão	Divide a primeira entrada pela segunda em um laço de subtrações	32, 8.	4	4	Sucesso
Potência	Eleva a primeira entrada à potência da segunda	4, 3.	64	64	Sucesso
Média	Retorna a média entre três valores de entrada	12, 18, 22.	17	17	Sucesso
Menor	Dadas três entradas, retornar qual das duas é a menor	3, 15,, 21.	3	3	Sucesso
Ordenação	Dadas quatro entradas, as ordena crescentemente.	5, 3, 54, 13	3, 5, 13, 54	3, 5, 13, 54	Sucesso
Teto e chão	Incrementa um valor até chegar à metade de um topo, o topo, o dobro do chão e o chão	—	9040, 18080, 100, 50.	9040, 18080, 100, 50.	Sucesso
Área do Círculo	Dada uma entrada, retorna a área de um círculo que tenha seu valor como raio.	4	51 (arredondado)	50 (truncado)	Sucesso (aproximado)
Multiplicação	Multiplica duas entradas através de um laço de somas	32, 8	256	256	Sucesso

Tabela 4 – Testes de multitarefa sobre Sistema Operacional.

Programa	**Teste**	**Resultado Esperado**	**Resultado Obtido**	**Resultado Final**
Multiplicação	8, 32.	256	64000	Sucesso
Potência	3, 4.	81	81	Sucesso
Fibonacci	9	34	34	Sucesso

A Tabela 4 apresenta os programas escolhidos para a execução e os resultados obtidos.

O sistema executou os programas em multitarefa com preempção com sucesso.

5.3 Testes de Operações com Arquivos

O sistema desenvolvido apresenta as operações sobre arquivo de *deletar*, *renomear*, *criar* e *editar*. Cada uma delas pôde ser realizada dentro da linha de comando.

O estado inicial da lista de arquivos (programas) é um vetor com 10 programas nomeados de 1 a 11. Abaixo se descrevem os testes realizados:

- Deletar:
  1. O arquivo ’13’ foi selecionado na lista de arquivos;
  2. A opção ’deletar’ foi selecionada;
  3. A ausência do arquivo ’13’ na listagem de arquivos e impossibilidade de selecioná-lo comprovaram sua deleção.
- Renomear:
  1. O arquivo ’16’ foi selecionado na lista de arquivos;
  2. A opção ’deletar’ foi selecionada;
  3. Entrou-se com o valor ’44’ como novo nome para o arquivo ’16’;
  4. A ausência do arquivo ’16’, a presença do arquivo ’44’ e a execução das instruções do antigo arquivo ’16’ através da execução do arquivo ’44’ comprovaram que este fora renomeado.
- Criar:
  1. A opção de criação foi selecionada na *prompt*;
  2. O ID ’21’ foi inserida para o novo arquivo;
  3. Sua presença na listagem de arquivos comprovou sua criação.
- Editar:

Tabela 5 – Instruções do teste de edição de arquivo.

Instrução	**Linha**	**Opcode**	**Registrador A**	**Registrador B**	**Registrador C**	**Imediato**
Input R[7]	0	29	7	0	0	0
Load Imediato R[9]	1	26	9	0	0	2
R[9]<- R[9]*R[7]	2	4	9	9	7	0
Output R[9]	3	32	9	0	0	0
Halt	4	28	0	0	0	0
Fim (linha<- 0)	5	0	0	0	0	0

1. As instruções contidas na Tabela 5 foram inseridas no arquivo '21';
2. A execução apropriada das instruções inseridas comprovou que a edição foi satisfatória.

6 Considerações Finais

O projeto de um sistema operacional toca os aspectos fundamentais do uso de computadores como os entendemos hoje em dia. Assim, ele traz uma bagagem de compreensão ao aluno muito abrangente e enriquecedora.

Os maiores desafios de implementação do projeto se devem à integração das diversas ferramentas, tanto as desenvolvidas pelo aluno quanto as convencionais.

O tamanho e complexidade do projeto trouxeram um grande aprendizado de desenvolvimento, tanto em planejamento quanto em organização, testes e implementação.

Espera-se que o presente projeto seja uma base sólida para o desenvolvimento do projeto posterior e integre um sistema computacional completo.

Referências

1 MORALES, D. *processor-galetron*. 2018. Webcite. Disponível em: <https://github.com/davimmorales/processor-galetron>. Acesso em: 25 fev. 2018. Citado 6 vezes nas páginas 2, 25, 26, 35, 36 e 39.

2 MORALES, D. *bird-whisperer-assembler*. 2018. Webcite. Disponível em: <https://github.com/davimmorales/bird-whisperer-assembler>. Acesso em: 25 fev. 2018. Citado 2 vezes nas páginas 2 e 25.

3 TANENBAUM, A. S. *Modern Operating Systems*. 3rd. ed. Upper Saddle River, NJ, USA: Prentice Hall Press, 2007. ISBN 9780136006633. Citado 13 vezes nas páginas 7, 8, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22 e 23.

4 ARPACI-DUSSEAU, R. H.; ARPACI-DUSSEAU, A. C. *Operating Systems: Three Easy Pieces*. 0.91. ed. [S.l.]: Arpaci-Dusseau Books, 2015. Citado na página 7.

5 KEIZER, G. *Windows comes up third in OS clash two years early*. 2016. Webcite. Disponível em: <https://www.computerworld.com/article/3050931/microsoft-windows/windows-comes-up-third-in-os-clash-two-years-early.html>. Acesso em: 30 set. 2017. Citado na página 7.

6 SILBERSCHATZ, A.; GALVIN, P. B.; GAGNE, G. *Operating System Concepts*. 8th. ed. [S.l.]: Wiley Publishing, 2008. ISBN 0470128720. Citado na página 8.

7 HOPE, C. *Race Conditions*. 2017. Webcite. Disponível em: <https://www.computerhope.com/jargon/r/race-condition.htm>. Acesso em: 30 set. 2017. Citado na página 16.

8 MORALES, D. *tucanOS / Trello*. 2018. Webcite. Disponível em: <https://trello.com/b/VhsGJNqh/tucanos>. Acesso em: 25 fev. 2018. Citado na página 25.

9 MORALES, D. *tucanOS-operating-system*. 2018. Webcite. Disponível em: <https://github.com/davimmorales/tucanOS-operating-system>. Acesso em: 25 fev. 2018. Citado na página 25.

Anexos

ANEXO A – Cógigos de Teste

A.1 Programa 0 - Fibonacci

```
int fibonacci(int n){
    int c;
    int next;
    int first;
    int second;
    first = 0;
    second = 1;
    c = 0;
    while(c <= n){
        if(c <= 1){
            next = c;
        }else{
            next = first + second;
            first = second;
            second = next; /* Estava second = first */
        }
        c = c + 1;
    }
    return next;
}
void main(void){
  int n;
  n = 3;
  output(fibonacci(n));
}
```

A.2 Programa 1 - Maior

```
int biggest(int arr[], int sz)
{
    int result;
    int ct;
    ct = 0;
    result = 0;

    while(ct<sz){
        if(arr[ct]>result){
                result = arr[ct];
        }
        ct = ct + 1;
   }

    return result;
}

void main(void)
{
    int array[3];
    int size;
    int count;
```

```
23
24      size = 3;
25      count = 0;
26      while(count<size){
27          array[count] = input();
28          count = count + 1;
29      }
30
31      output(biggest(array,size));
32  }
```

A.3 Programa 2 - Divisão

```
1   #include "stdio.h"
2
3   int main(int argc, char const *argv[]) {
4     int a;
5     int b;
6     int c;
7     c = 0;
8     scanf("%d", &a);
9     scanf("%d", &b);
10    while (b<a+1) {
11      a = a - b;
12      c = c + 1;
13    }
14    printf("%d\n", c);
15    return 0;
16  }
```

A.4 Programa 3 - Potência

```
1   int pow(int x, int y) {
2       int result;
3       result = 1;
4       while(0<b){
5         result = result*a;
6         b = b - 1;
7       }
8   }
9
10  void main(void) {
11      int x;
12      int y;
13
14      x = input();
15      y = input();
16      output(pow(x, y));
17  }
```

A.5 Programa 4 - Média

```
1   int mean(int arr[], int sz)
2   {
3       int result;
4       int ct;
```

```
5       result = 0;
6       while(ct<sz){
7           result = result + arr[ct];
8           ct = ct + 1;
9      }
10      result = result/sz;
11      return result;
12  }
13
14  void main(void)
15  {
16      int array[3];
17      int size;
18      int count;
19
20      size = 3;
21      count = 0;
22      while(count<size){
23          array[count] = input();
24          count = count + 1;
25      }
26
27      output(mean(array,size));
28  }
```

A.6 Programa 5 - Menor

```
1   int smallest(int arr[], int sz)
2   {
3       int result;
4       int ct;
5       ct = 0;
6       result = 250000;
7
8       while(ct<sz){
9           if(arr[ct]<result){
10                  result = arr[ct];
11          }
12          ct = ct + 1;
13     }
14
15      return result;
16  }
17
18  void main(void)
19  {
20      int array[3];
21      int size;
22      int count;
23
24      size = 3;
25      count = 0;
26      while(count<size){
27          array[count] = input();
28          count = count + 1;
29      }
30
31      output(smallest(array,size));
32  }
```

A.7 Programa 6 - Ordenação

```
1  void sort(int num[], int tam){
2      int i;
3      int j;
4      int min;
5      int aux;
6      i = 0;
7      while (i < tam-1){
8          min = i;
9          j = i + 1;
10         while (j < tam){
11             if(num[j] < num[min])
12                 min = j;
13             j = j + 1;
14         }
15         if (i != min){
16             aux = num[i];
17             num[i] = num[min];
18             num[min] = aux;
19         }
20         i = i + 1;
21     }
22 }
23 void main(void){
24     int vetor[4];
25     int i;
26     vetor[0] = 9;
27     vetor[1] = 6;
28     vetor[2] = 8;
29     vetor[3] = 7;
30     sort(vetor, 4);
31     i = 1;
32     output(vetor[i]);
33 }
```

A.8 Programa 7 - Multiplicação

```
1  void main(void){
2    int a;
3    int b;
4    int result;
5    input(a);
6    input(b);
7    result = 0;
8
9    while (0<b) {
10     result = result + a;
11     b = b - 1;
12   }
13   output(result);
14 }
```

A.9 Programa 8 - Área do Círculo

A.10 Programa 9 - Teto e Chão

```
1  void main(void){
2    int top;
3    int count;
4    int bottom;
5    top = 18080;
6    bottom = 50;
7    count = 0;
8    while (count<top) {
9      if (count==top/2) {
10       output(count);
11     }
12     count = count + 1;
13   }
14
15   while (bottom<count) {
16     if (count==bottom*2) {
17       output(count);
18     }
19     count = count - 1;
20   }
21
22 }
```

ANEXO B – Código Fonte do Sistema Operacional

B.1 Código Fonte do SO em *assembly*

```
jump 0 0 0 257
input 21 0 0 0
loadi 22 0 0 0
setlt 21 22 23 0
setlt 22 21 24 0
or 23 24 23 0
pbranch 23 0 0 0
branchz 0 0 0 19
loadi 22 0 0 1
setlt 21 22 23 0
setlt 22 21 24 0
or 23 24 23 0
pbranch 23 0 0 0
branchz 0 0 0 16
loadi 22 0 0 2
setlt 21 22 23 0
setlt 22 21 24 0
or 23 24 23 0
pbranch 23 0 0 0
branchz 0 0 0 23
noo 0 0 0 0
noo 0 0 0 0
noo 0 0 0 0
noo 0 0 0 0
noo 0 0 0 0
noo 0 0 0 0
jump 0 0 0 855
loadi 22 0 0 0
output 22 0 0 0
jump 0 0 0 257
loadi 21 0 0 0
loadi 22 0 0 192
loadr 22 0 23 0
setlt 21 23 24 0
setlt 23 21 25 0
or 24 25 25 0
not 25 0 25 0
pbranch 25 0 0 0
branchz 0 0 0 1
jump 0 0 0 257
output 23 0 0 0
addi 22 0 22 2
jump 0 0 0 288
input 21 0 0 0
loadi 22 0 0 0
setlt 21 22 23 0
setlt 22 21 24 0
or 23 24 23 0
```

```
49  pbranch 23 0 0 0
50  branchz 0 0 0 13
51  loadi 22 0 0 1
52  setlt 21 22 23 0
53  setlt 22 21 24 0
54  or 23 24 23 0
55  pbranch 23 0 0 0
56  branchz 0 0 0 11
57  loadi 22 0 0 2
58  setlt 21 22 23 0
59  setlt 22 21 24 0
60  or 23 24 23 0
61  pbranch 23 0 0 0
62  branchz 0 0 0 4
63  jump 0 0 0 343
64  loadi 22 0 0 1
65  output 22 0 0 0
66  jump 0 0 0 299
67  jump 0 0 0 257
68  loadi 21 0 0 0
69  loadi_hd 22 1 0 0
70  load_hd 22 0 23 0
71  setlt 21 23 24 0
72  setlt 23 21 25 0
73  or 24 25 25 0
74  not 25 0 25 0
75  pbranch 25 0 0 0
76  branchz 0 0 0 1
77  jump 0 0 0 299
78  loadi 21 0 0 1
79  setlt 21 23 24 0
80  setlt 23 21 25 0
81  or 24 25 25 0
82  pbranch 25 0 0 0
83  branchz 0 0 0 1
84  output 23 0 0 0
85  addi 22 0 22 32
86  loadi 21 0 0 0
87  jump 0 0 0 325
88  loadi 22 0 0 3
89  setlt 21 22 23 0
90  setlt 22 21 24 0
91  or 23 24 23 0
92  pbranch 23 0 0 0
93  branchz 0 0 0 1083
94  loadi 0 0 0 0
95  loadi 22 0 0 192
96  loadr 22 0 23 0
97  setlt 0 23 24 0
98  setlt 23 0 25 0
99  or 24 25 25 0
100 pbranch 25 0 0 0
101 branchz 0 0 0 9
102 setlt 21 23 24 0
103 setlt 23 21 25 0
104 or 24 25 25 0
105 not 25 0 25 0
106 pbranch 25 0 0 0
107 branchz 0 0 0 1
108 jump 0 0 0 299
```

```
109 addi 22 0 22 2
110 jump 0 0 0 351
111 loadi_hd 22 1 0 0
112 load_hd 22 0 23 0
113 setlt 0 23 24 0
114 setlt 23 0 25 0
115 or 24 25 25 0
116 not 25 0 25 0
117 pbranch 25 0 0 0
118 branchz 0 0 0 1
119 jump 0 0 0 299
120 setlt 21 23 24 0
121 setlt 23 21 25 0
122 or 24 25 25 0
123 pbranch 25 0 0 0
124 branchz 0 0 0 2
125 addi 22 0 22 32
126 jump 0 0 0 367
127 store 21 0 0 234
128 input 21 0 0 0
129 loadi 0 0 0 0
130 setlt 21 0 23 0
131 setlt 0 21 24 0
132 or 23 24 23 0
133 not 23 0 23 0
134 pbranch 23 0 0 0
135 branchz 0 0 0 3
136 load 22 0 0 234
137 output 22 0 0 0
138 jump 0 0 0 383
139 loadi 22 0 0 2
140 setlt 21 22 23 0
141 setlt 22 21 24 0
142 or 23 24 23 0
143 not 23 0 23 0
144 pbranch 23 0 0 0
145 branchz 0 0 0 1
146 jump 0 0 0 299
147 jump 0 0 0 606
148 loadi 27 0 0 582
149 jump 0 0 0 405
150 loadi 0 0 0 0
151 store 0 0 0 224
152 store 0 0 0 226
153 loadi 21 0 0 192
154 store 21 0 0 227
155 load 21 0 0 227
156 loadr 21 0 22 0
157 setlt 22 0 23 0
158 setlt 0 22 24 0
159 or 23 24 23 0
160 pbranch 23 0 0 0
161 branchz 0 0 0 20
162 loadi 24 0 0 1
163 setlt 24 22 24 0
164 pbranch 24 0 0 0
165 branchz 0 0 0 12
166 load 21 0 0 227
167 addi 21 0 21 1
168 loadr 21 0 22 0
```

```
169 setlt 22 0 23 0
170 setlt 0 22 24 0
171 or 23 24 23 0
172 not 23 0 23 0
173 pbranch 23 0 0 0
174 branchz 0 0 0 3
175 load 24 0 0 226
176 addi 24 0 24 1
177 store 24 0 0 226
178 load 21 0 0 227
179 addi 21 0 21 2
180 store 21 0 0 227
181 jump 0 0 0 410
182 load 21 0 0 226
183 loadi 22 0 0 2
184 setlt 22 21 22 0
185 pbranch 22 0 0 0
186 branchz 0 0 0 2
187 loadi 21 0 0 1
188 store 21 0 0 224
189 noo 0 0 0 0
190 noo 0 0 0 0
191 noo 0 0 0 0
192 noo 0 0 0 0
193 noo 0 0 0 0
194 jumpr 27 0 0 0
195 load 21 0 0 234
196 store 21 0 0 271
197 loadi 27 0 0 496
198 load 21 0 0 234
199 store 21 0 0 228
200 store 0 0 0 229
201 store 0 0 0 230
202 loadi 21 0 0 1
203 store 21 0 0 231
204 store 0 0 0 232
205 load 30 0 0 271
206 loadi_hd 21 1 0 0
207 store 21 0 0 233
208 load 21 0 0 233
209 load_hd 21 0 22 0
210 load 23 0 0 228
211 setlt 22 23 24 0
212 setlt 23 22 25 0
213 or 24 25 24 0
214 pbranch 24 0 0 0
215 branchz 0 0 0 2
216 addi 21 0 21 32
217 jump 0 0 0 462
218 load 21 0 0 233
219 addi 21 0 21 1
220 load 22 0 0 229
221 store_hd 21 0 22 0
222 addi 21 0 21 1
223 load 22 0 0 230
224 store_hd 21 0 22 0
225 addi 21 0 21 1
226 load 22 0 0 231
227 store_hd 21 0 22 0
228 addi 21 0 21 1
```

```
229 load 22 0 0 232
230 store_hd 21 0 22 0
231 noo 0 0 0 0
232 noo 0 0 0 0
233 noo 0 0 0 0
234 noo 0 0 0 0
235 noo 0 0 0 0
236 noo 0 0 0 0
237 noo 0 0 0 0
238 noo 0 0 0 0
239 noo 0 0 0 0
240 jumpr 27 0 0 0
241 loadi 27 0 0 526
242 load 21 0 0 234
243 store 21 0 0 235
244 store 0 0 0 236
245 jump 0 0 0 501
246 loadi 21 0 0 192
247 loadi 22 0 0 1
248 loadr 21 0 24 0
249 setlt 22 24 23 0
250 pbranch 23 0 0 0
251 branchz 0 0 0 2
252 addi 21 0 21 2
253 jump 0 0 0 503
254 load 22 0 0 235
255 load 23 0 0 236
256 storer 21 0 22 0
257 addi 21 0 21 1
258 storer 21 0 23 0
259 noo 0 0 0 0
260 noo 0 0 0 0
261 noo 0 0 0 0
262 noo 0 0 0 0
263 noo 0 0 0 0
264 noo 0 0 0 0
265 noo 0 0 0 0
266 noo 0 0 0 0
267 noo 0 0 0 0
268 noo 0 0 0 0
269 noo 0 0 0 0
270 jumpr 27 0 0 0
271 load 21 0 0 234
272 store 21 0 0 241
273 loadi 27 0 0 555
274 noo 0 0 0 0
275 loadi_hd 22 1 0 0
276 store 22 0 0 243
277 load 21 0 0 243
278 load_hd 21 0 22 0
279 setlt 0 22 23 0
280 setlt 22 0 24 0
281 or 23 24 23 0
282 pbranch 23 0 0 0
283 branchz 0 0 0 14
284 load 25 0 0 241
285 setlt 22 25 23 0
286 setlt 25 22 24 0
287 or 23 24 23 0
288 not 23 0 23 0
```

```
289 pbranch 23 0 0 0
290 branchz 0 0 0 4
291 addi 21 0 22 5
292 load_hd 22 0 22 0
293 store 22 0 0 242
294 jump 0 0 0 554
295 addi 21 0 21 32
296 store 21 0 0 243
297 jump 0 0 0 532
298 store 0 0 0 242
299 jumpr 27 0 0 0
300 load 21 0 0 242
301 store 21 0 0 238
302 loadi 27 0 0 581
303 store 0 0 0 239
304 loadi 21 0 0 192
305 load 22 0 0 238
306 times 21 22 21 0
307 noo 0 0 0 0
308 loadi_hd 22 2 0 0
309 add 21 22 21 0
310 store 21 0 0 240
311 load 21 0 0 240
312 load 22 0 0 239
313 add 21 22 21 0
314 load_hd 21 0 22 0
315 setlt 22 0 23 0
316 setlt 0 22 24 0
317 or 23 24 23 0
318 pbranch 23 0 0 0
319 branchz 0 0 0 5
320 load 23 0 0 239
321 store_i_ram 23 0 22 0
322 addi 23 0 23 1
323 store 23 0 0 239
324 jump 0 0 0 566
325 jumpr 27 0 0 0
326 jump 0 0 0 0
327 loadi 27 0 0 598
328 jump 0 0 0 584
329 store 0 0 0 244
330 loadi 22 0 0 1
331 loadi 21 0 0 192
332 loadr 21 0 23 0
333 setlt 22 23 24 0
334 pbranch 24 0 0 0
335 branchz 0 0 0 4
336 loadi 25 0 0 1
337 store 25 0 0 244
338 addi 21 0 21 2
339 jump 0 0 0 587
340 noo 0 0 0 0
341 noo 0 0 0 0
342 jumpr 27 0 0 0
343 load 21 0 0 244
344 setlt 21 0 22 0
345 setlt 0 21 23 0
346 or 22 23 22 0
347 not 22 0 22 0
348 pbranch 22 0 0 0
```

```
branchz 0 0 0 27
jump 0 0 0 450
loadi 22 0 0 1
setlt 21 22 23 0
setlt 22 21 24 0
or 23 24 23 0
not 23 0 23 0
pbranch 23 0 0 0
branchz 0 0 0 1
jump 0 0 0 582
loadi 22 0 0 3
setlt 21 22 23 0
setlt 22 21 24 0
or 23 24 23 0
pbranch 23 0 0 0
branchz 0 0 0 754
loadi 22 0 0 4
setlt 21 22 23 0
setlt 22 21 24 0
or 23 24 23 0
pbranch 23 0 0 0
branchz 0 0 0 782
loadi 22 0 0 5
setlt 21 22 23 0
setlt 22 21 24 0
or 23 24 23 0
pbranch 23 0 0 0
branchz 0 0 0 870
loadi 27 0 0 634
noo 0 0 0 0
noo 0 0 0 0
noo 0 0 0 0
noo 0 0 0 0
noo 0 0 0 0
noo 0 0 0 0
noo 0 0 0 0
noo 0 0 0 0
noo 0 0 0 0
loadi 27 0 0 678
loadi_hd 21 1 0 0
store 21 0 0 246
load 21 0 0 246
load_hd 21 0 22 0
setlt 0 22 23 0
pbranch 23 0 0 0
branchz 0 0 0 20
addi 21 0 21 1
load_hd 21 0 23 0
setlt 0 23 25 0
not 25 0 25 0
pbranch 25 0 0 0
branchz 0 0 0 11
addi 21 0 21 2
load_hd 21 0 23 0
loadi 24 0 0 1
setlt 24 23 25
setlt 23 24 24
or 25 24 25 0
not 25 0 25 0
pbranch 25 0 0 0
```

```
409 branchz 0 0 0 2
410 store 22 0 0 247
411 jump 0 0 0 670
412 load 21 0 0 246
413 addi 21 0 21 32
414 jump 0 0 0 644
415 noo 0 0 0 0
416 noo 0 0 0 0
417 noo 0 0 0 0
418 noo 0 0 0 0
419 noo 0 0 0 0
420 noo 0 0 0 0
421 noo 0 0 0 0
422 jumpr 27 0 0 0
423 loadi 27 0 0 737
424 noo 0 0 0 0
425 loadi 21 0 0 3
426 store 21 0 0 249
427 store 0 0 0 266
428 store 0 0 0 267
429 store 0 0 0 268
430 loadi_hd 21 1 0 0
431 store 21 0 0 248
432 load 21 0 0 248
433 load_hd 21 0 22 0
434 setlt 0 22 23 0
435 pbranch 23 0 0 0
436 branchz 0 0 0 20
437 loadi 23 0 0 1
438 setlt 23 22 24 0
439 pbranch 24 0 0 0
440 branchz 0 0 0 13
441 addi 21 0 21 3
442 load_hd 21 0 22 0
443 setlt 22 23 24 0
444 noo 0 0 0 0
445 noo 0 0 0 0
446 not 24 0 24 0
447 pbranch 24 0 0 0
448 branchz 0 0 0 5
449 subi 21 0 21 2
450 load_hd 21 0 22 0
451 loadi 24 0 0 266
452 add 22 24 22 0
453 storer 22 0 23 0
454 load 21 0 0 248
455 addi 21 0 21 32
456 jump 0 0 0 686
457 load 21 0 0 267
458 setlt 0 21 22 0
459 not 22 0 22 0
460 pbranch 22 0 0 0
461 branchz 0 0 0 3
462 loadi 23 0 0 1
463 store 23 0 0 249
464 jump 0 0 0 727
465 load 21 0 0 268
466 setlt 0 21 22 0
467 not 22 0 22 0
468 pbranch 22 0 0 0
```

```
469 branchz 0 0 0 2
470 loadi 23 0 0 2
471 store 23 0 0 249
472 noo 0 0 0 0
473 noo 0 0 0 0
474 noo 0 0 0 0
475 noo 0 0 0 0
476 noo 0 0 0 0
477 noo 0 0 0 0
478 noo 0 0 0 0
479 noo 0 0 0 0
480 noo 0 0 0 0
481 jumpr 27 0 0 0
482 load 21 0 0 247
483 store 21 0 0 250
484 load 21 0 0 249
485 store 21 0 0 251
486 loadi 27 0 0 788
487 noo 0 0 0 0
488 loadi_hd 21 1 0 0
489 store 21 0 0 252
490 load 21 0 0 252
491 load_hd 21 0 22 0
492 setlt 0 22 23 0
493 pbranch 23 0 0 0
494 branchz 0 0 0 16
495 load 23 0 0 250
496 setlt 22 23 24 0
497 setlt 23 22 23 0
498 or 23 24 23 0
499 not 23 0 23 0
500 pbranch 23 0 0 0
501 branchz 0 0 0 6
502 addi 21 0 21 1
503 load_hd 21 0 23 0
504 store 23 0 0 253
505 load 23 0 0 251
506 store_hd 21 0 23 0
507 jump 0 0 0 766
508 load 21 0 0 252
509 addi 21 0 21 32
510 jump 0 0 0 744
511 load 22 0 0 253
512 loadi 23 0 0 38
513 times 22 23 22 0
514 store 22 0 0 253
515 load 22 0 0 251
516 times 22 23 22 0
517 store 22 0 0 254
518 loadi 21 0 0 0
519 store 21 0 0 252
520 load 21 0 0 252
521 setlt 21 23 24 0
522 pbranch 24 0 0 0
523 branchz 0 0 0 8
524 load 22 0 0 253
525 add 21 22 22 0
526 loadr 22 0 22 0
527 load 24 0 0 254
528 add 21 24 24 0
```

```
529 storer 24 0 22 0
530 addi 21 0 21 1
531 jump 0 0 0 774
532 jumpr 27 0 0 0
533 load 21 0 0 247
534 store 21 0 0 255
535 loadi 27 0 0 450
536 output 20 0 0 0
537 loadi_hd 21 1 0 0
538 store 21 0 0 256
539 load 21 0 0 256
540 load_hd 21 0 22 0
541 setlt 0 22 23 0
542 pbranch 23 0 0 0
543 branchz 0 0 0 11
544 load 23 0 0 255
545 setlt 22 23 24 0
546 setlt 23 22 25 0
547 or 24 25 24 0
548 not 24 0 24 0
549 pbranch 24 0 0 0
550 branchz 0 0 0 1
551 jump 0 0 0 810
552 load 21 0 0 256
553 addi 21 0 21 32
554 jump 0 0 0 793
555 store 21 0 0 256
556 load 21 0 0 256
557 addi 21 0 21 4
558 store_hd 21 0 28 0
559 addi 21 0 21 8
560 store_hd 21 0 0 0
561 addi 21 0 21 1
562 store_hd 21 0 1 0
563 addi 21 0 21 1
564 store_hd 21 0 2 0
565 addi 21 0 21 1
566 store_hd 21 0 3 0
567 addi 21 0 21 1
568 store_hd 21 0 4 0
569 addi 21 0 21 1
570 store_hd 21 0 5 0
571 addi 21 0 21 1
572 store_hd 21 0 6 0
573 addi 21 0 21 1
574 store_hd 21 0 7 0
575 addi 21 0 21 1
576 store_hd 21 0 8 0
577 addi 21 0 21 1
578 store_hd 21 0 9 0
579 addi 21 0 21 1
580 store_hd 21 0 10 0
581 addi 21 0 21 1
582 store_hd 21 0 11 0
583 addi 21 0 21 1
584 store_hd 21 0 12 0
585 addi 21 0 21 1
586 store_hd 21 0 13 0
587 addi 21 0 21 1
588 store_hd 21 0 14 0
```

```
589 addi 21 0 21 1
590 store_hd 21 0 15 0
591 addi 21 0 21 1
592 store_hd 21 0 16 0
593 addi 21 0 21 1
594 store_hd 21 0 17 0
595 addi 21 0 21 1
596 store_hd 21 0 18 0
597 addi 21 0 21 1
598 store_hd 21 0 19 0
599 jumpr 27 0 0 0
600 loadi 22 0 0 10
601 setlt 21 22 23 0
602 pbranch 23 0 0 0
603 branchz 0 0 0 1
604 jump 0 0 0 1245
605 store 21 0 0 260
606 store 21 0 0 257
607 loadi 27 0 0 895
608 jump 0 0 0 864
609 store 0 0 0 259
610 loadi_hd 21 1 0 0
611 store 21 0 0 258
612 load 21 0 0 258
613 load_hd 21 0 22 0
614 setlt 0 22 23 0
615 pbranch 23 0 0 0
616 branchz 0 0 0 21
617 load 23 0 0 257
618 setlt 22 23 24 0
619 setlt 23 22 23 0
620 or 23 24 23 0
621 not 23 0 23 0
622 pbranch 23 0 0 0
623 branchz 0 0 0 11
624 addi 21 0 21 3
625 load_hd 21 0 22 0
626 loadi 23 0 0 1
627 setlt 22 23 24 0
628 noo 0 0 0 0
629 noo 0 0 0 0
630 not 24 0 24 0
631 pbranch 24 0 0 0
632 branchz 0 0 0 2
633 store 23 0 0 259
634 jump 0 0 0 893
635 load 21 0 0 258
636 addi 21 0 21 32
637 jump 0 0 0 866
638 noo 0 0 0 0
639 jumpr 27 0 0 0
640 load 21 0 0 259
641 setlt 0 21 22 0
642 not 22 0 22 0
643 pbranch 22 0 0 0
644 branchz 0 0 0 1
645 jump 0 0 0 257
646 input 21 0 0 0
647 setlt 0 21 22 0
648 not 22 0 22 0
```

```
649 pbranch 22 0 0 0
650 branchz 0 0 0 3
651 load 22 0 0 260
652 output 22 0 0 0
653 jump 0 0 0 901
654 loadi 24 0 0 2
655 setlt 21 24 22 0
656 setlt 24 21 23 0
657 or 22 23 22 0
658 not 22 0 22 0
659 pbranch 22 0 0 0
660 branchz 0 0 0 1
661 jump 0 0 0 257
662 loadi 24 0 0 3
663 setlt 21 24 22 0
664 setlt 24 21 23 0
665 or 22 23 22 0
666 not 22 0 22 0
667 pbranch 22 0 0 0
668 branchz 0 0 0 1
669 jump 0 0 0 1355
670 setlt 24 21 22 0
671 pbranch 22 0 0 0
672 branchz 0 0 0 1
673 jump 0 0 0 901
674 loadi 24 0 0 1
675 setlt 21 24 22 0
676 setlt 24 21 23 0
677 or 22 23 22 0
678 pbranch 22 0 0 0
679 branchz 0 0 0 1
680 jump 0 0 0 901
681 load 21 0 0 260
682 store 21 0 0 261
683 store 21 0 0 271
684 loadi 27 0 0 1347
685 load 30 0 0 271
686 loadi_hd 21 1 0 0
687 store 21 0 0 262
688 load 21 0 0 262
689 load_hd 21 0 22 0
690 setlt 0 22 23 0
691 pbranch 23 0 0 0
692 branchz 0 0 0 20
693 loadi 24 0 0 1
694 setlt 22 24 23 0
695 setlt 24 22 24 0
696 or 24 23 24 0
697 pbranch 24 0 0 0
698 branchz 0 0 0 11
699 load 24 0 0 261
700 setlt 22 24 23 0
701 setlt 24 22 24 0
702 or 24 23 24 0
703 not 24 0 24 0
704 pbranch 24 0 0 0
705 branchz 0 0 0 4
706 addi 21 0 21 1
707 load_hd 21 0 24 0
708 store 24 0 0 263
```

```
709 jump 0 0 0 968
710 load 21 0 0 262
711 addi 21 0 21 32
712 jump 0 0 0 942
713 noo 0 0 0 0
714 noo 0 0 0 0
715 noo 0 0 0 0
716 noo 0 0 0 0
717 jumpr 27 0 0 0
718 load 21 0 0 260
719 store 21 0 0 250
720 loadi 21 0 0 4
721 store 21 0 0 251
722 loadi 27 0 0 979
723 jump 0 0 0 743
724 loadi 27 0 0 981
725 jump 0 0 0 643
726 load 21 0 0 247
727 store 21 0 0 250
728 load 21 0 0 263
729 store 21 0 0 251
730 loadi 27 0 0 987
731 jump 0 0 0 743
732 load 21 0 0 260
733 store 21 0 0 250
734 store 0 0 0 251
735 loadi 27 0 0 992
736 jump 0 0 0 743
737 load 21 0 0 247
738 store 21 0 0 255
739 loadi 27 0 0 996
740 jump 0 0 0 792
741 load 21 0 0 260
742 store 21 0 0 264
743 loadi 27 0 0 1067
744 jump 0 0 0 1000
745 loadi_hd 21 1 0 0
746 store 21 0 0 265
747 load 21 0 0 265
748 load_hd 21 0 22 0
749 setlt 0 22 23 0
750 pbranch 23 0 0 0
751 branchz 0 0 0 11
752 load 23 0 0 264
753 setlt 22 23 24 0
754 setlt 23 22 25 0
755 or 24 25 24 0
756 not 24 0 24 0
757 pbranch 24 0 0 0
758 branchz 0 0 0 1
759 jump 0 0 0 1018
760 load 21 0 0 265
761 addi 21 0 21 32
762 jump 0 0 0 1001
763 store 21 0 0 265
764 load 21 0 0 265
765 addi 21 0 21 4
766 load_hd 21 0 28 0
767 addi 21 0 21 8
768 load_hd 21 0 0 0
```

```
addi 21 0 21 1
load_hd 21 0 1 0
addi 21 0 21 1
load_hd 21 0 2 0
addi 21 0 21 1
load_hd 21 0 3 0
addi 21 0 21 1
load_hd 21 0 4 0
addi 21 0 21 1
load_hd 21 0 5 0
addi 21 0 21 1
load_hd 21 0 6 0
addi 21 0 21 1
load_hd 21 0 7 0
addi 21 0 21 1
load_hd 21 0 8 0
addi 21 0 21 1
load_hd 21 0 9 0
addi 21 0 21 1
load_hd 21 0 10 0
addi 21 0 21 1
load_hd 21 0 11 0
addi 21 0 21 1
load_hd 21 0 12 0
addi 21 0 21 1
load_hd 21 0 13 0
addi 21 0 21 1
load_hd 21 0 14 0
addi 21 0 21 1
load_hd 21 0 15 0
addi 21 0 21 1
load_hd 21 0 16 0
addi 21 0 21 1
load_hd 21 0 17 0
addi 21 0 21 1
load_hd 21 0 18 0
addi 21 0 21 1
load_hd 21 0 19 0
noo 0 0 0 0
noo 0 0 0 0
noo 0 0 0 0
noo 0 0 0 0
jumpr 27 0 0 0
loadi 27 0 0 1071
load 21 0 0 260
store 21 0 0 241
jump 0 0 0 530
load 21 0 0 242
store 21 0 0 238
loadi 27 0 0 1075
jump 0 0 0 558
jumpr 28 0 0 0
noo 0 0 0 0
noo 0 0 0 0
noo 0 0 0 0
noo 0 0 0 0
noo 0 0 0 0
noo 0 0 0 0
noo 0 0 0 0
loadi 27 0 0 257
```

```
829 jump 0 0 0 1085
830 load_wd 21 0 0 0
831 store 21 0 0 270
832 load 21 0 0 270
833 setlt 0 21 22 0
834 pbranch 22 0 0 0
835 branchz 0 0 0 15
836 loadi 23 0 0 4
837 setlt 21 23 22 0
838 pbranch 22 0 0 0
839 branchz 0 0 0 1
840 jump 0 0 0 1108
841 loadi 23 0 0 5
842 setlt 21 23 22 0
843 pbranch 22 0 0 0
844 branchz 0 0 0 1
845 jump 0 0 0 1212
846 loadi 23 0 0 6
847 setlt 21 23 22 0
848 pbranch 22 0 0 0
849 branchz 0 0 0 1
850 jump 0 0 0 1284
851 noo 0 0 0 0
852 jumpr 27 0 0 0
853 noo 0 0 0 0
854 noo 0 0 0 0
855 noo 0 0 0 0
856 noo 0 0 0 0
857 noo 0 0 0 0
858 loadi 27 0 0 1155
859 load 21 0 0 270
860 store 21 0 0 272
861 jump 0 0 0 1117
862 store 0 0 0 274
863 loadi_hd 21 1 0 0
864 store 21 0 0 273
865 load 21 0 0 273
866 load_hd 21 0 22 0
867 setlt 0 22 23 0
868 pbranch 23 0 0 0
869 branchz 0 0 0 26
870 loadi 23 0 0 1
871 setlt 23 22 23 0
872 pbranch 23 0 0 0
873 branchz 0 0 0 19
874 addi 21 0 21 3
875 load_hd 21 0 23 0
876 setlt 0 23 24 0
877 pbranch 24 0 0 0
878 branchz 0 0 0 14
879 loadi 24 0 0 3
880 setlt 23 24 24 0
881 pbranch 24 0 0 0
882 branchz 0 0 0 10
883 subi 21 0 21 2
884 load_hd 21 0 23 0
885 load 24 0 0 272
886 setlt 23 24 21 0
887 setlt 24 23 24 0
888 or 21 24 24 0
```

```
889 not 24 0 24 0
890 pbranch 24 0 0 0
891 branchz 0 0 0 1
892 store 22 0 0 274
893 load 21 0 0 273
894 addi 21 0 21 32
895 jump 0 0 0 1119
896 noo 0 0 0 0
897 noo 0 0 0 0
898 noo 0 0 0 0
899 jumpr 27 0 0 0
900 noo 0 0 0 0
901 noo 0 0 0 0
902 load 21 0 0 274
903 setlt 0 21 22 0
904 not 22 0 22 0
905 pbranch 22 0 0 0
906 branchz 0 0 0 1
907 jumpr 28 0 0 0
908 noo 0 0 0 0
909 noo 0 0 0 0
910 noo 0 0 0 0
911 load 21 0 0 274
912 store 21 0 0 275
913 loadi 27 0 0 1194
914 jump 0 0 0 1170
915 store 0 0 0 277
916 loadi_hd 21 1 0 0
917 store 21 0 0 276
918 load 21 0 0 276
919 load_hd 21 0 22 0
920 setlt 0 22 23 0
921 pbranch 23 0 0 0
922 branchz 0 0 0 13
923 load 23 0 0 275
924 setlt 22 23 24
925 setlt 23 22 23
926 or 23 24 23 0
927 not 23 0 23 0
928 pbranch 23 0 0 0
929 branchz 0 0 0 3
930 addi 21 0 21 3
931 load_hd 21 0 23 0
932 store 23 0 0 277
933 load 21 0 0 276
934 addi 21 0 21 32
935 jump 0 0 0 1172
936 noo 0 0 0 0
937 noo 0 0 0 0
938 jumpr 27 0 0 0
939 noo 0 0 0 0
940 noo 0 0 0 0
941 load 21 0 0 277
942 loadi 22 0 0 2
943 setlt 21 22 23 0
944 setlt 22 21 22 0
945 or 22 23 22 0
946 not 22 0 22 0
947 pbranch 22 0 0 0
948 branchz 0 0 0 1
```

```
949  jumpr 28 0 0 0
950  load 21 0 0 274
951  store 21 0 0 261
952  store 21 0 0 260
953  loadi 27 0 0 973
954  jump 0 0 0 941
955  noo 0 0 0 0
956  noo 0 0 0 0
957  noo 0 0 0 0
958  noo 0 0 0 0
959  loadi 27 0 0 1216
960  jump 0 0 0 643
961  load 21 0 0 271
962  load 22 0 0 247
963  setlt 21 22 23 0
964  setlt 22 21 24 0
965  or 23 24 23 0
966  not 23 0 23 0
967  pbranch 23 0 0 0
968  branchz 0 0 0 3
969  noo 0 0 0 0
970  jumpr 28 0 0 0
971  noo 0 0 0 0
972  noo 0 0 0 0
973  noo 0 0 0 0
974  loadi 27 0 0 1238
975  load 21 0 0 247
976  store 21 0 0 228
977  store 0 0 0 229
978  store 0 0 0 230
979  loadi 21 0 0 2
980  store 21 0 0 231
981  store 28 0 0 232
982  jump 0 0 0 461
983  noo 0 0 0 0
984  noo 0 0 0 0
985  load 21 0 0 271
986  store 21 0 0 261
987  store 21 0 0 260
988  loadi 27 0 0 973
989  jump 0 0 0 941
990  noo 0 0 0 0
991  noo 0 0 0 0
992  noo 0 0 0 0
993  noo 0 0 0 0
994  store 21 0 0 279
995  loadi 22 0 0 4
996  setlt 21 22 23 0
997  pbranch 23 0 0 0
998  branchz 0 0 0 26
999  loadi 27 0 0 1277
1000 jump 0 0 0 1256
1001 loadi_hd 21 1 0 0
1002 store 21 0 0 278
1003 load 21 0 0 278
1004 load_hd 21 0 22 0
1005 setlt 0 22 23 0
1006 pbranch 23 0 0 0
1007 branchz 0 0 0 11
1008 addi 21 0 21 3
```

```
1009 load_hd 21 0 23 0
1010 loadi 21 0 0 2
1011 setlt 23 21 24 0
1012 not 24 0 24 0
1013 pbranch 24 0 0 0
1014 branchz 0 0 0 1
1015 output 22 0 0 0
1016 load 21 0 0 278
1017 addi 21 0 21 32
1018 jump 0 0 0 1257
1019 noo 0 0 0 0
1020 noo 0 0 0 0
1021 jumpr 27 0 0 0
1022 noo 0 0 0 0
1023 loadi 21 0 0 0
1024 jump 0 0 0 257
1025 noo 0 0 0 0
1026 loadi 21 0 0 0
1027 jump 0 0 0 257
1028 noo 0 0 0 0
1029 noo 0 0 0 0
1030 noo 0 0 0 0
1031 noo 0 0 0 0
1032 noo 0 0 0 0
1033 loadi 27 0 0 1290
1034 jump 0 0 0 643
1035 load 21 0 0 247
1036 store 21 0 0 228
1037 store 0 0 0 229
1038 store 0 0 0 230
1039 store 0 0 0 231
1040 store 0 0 0 232
1041 loadi 27 0 0 1298
1042 jump 0 0 0 461
1043 noo 0 0 0 0
1044 noo 0 0 0 0
1045 noo 0 0 0 0
1046 noo 0 0 0 0
1047 loadi 27 0 0 1328
1048 load 21 0 0 247
1049 store 21 0 0 280
1050 jump 0 0 0 1306
1051 loadi 21 0 0 192
1052 store 21 0 0 281
1053 load 21 0 0 281
1054 loadr 21 0 22 0
1055 setlt 0 22 23 0
1056 pbranch 23 0 0 0
1057 branchz 0 0 0 12
1058 load 23 0 0 280
1059 setlt 22 23 24 0
1060 setlt 23 22 23 0
1061 or 23 24 23 0
1062 not 23 0 23 0
1063 pbranch 23 0 0 0
1064 branchz 0 0 0 2
1065 loadi 23 0 0 1
1066 storer 21 0 23 0
1067 load 21 0 0 281
1068 addi 21 0 21 2
```

```
1069 jump 0 0 0 1307
1070 noo 0 0 0 0
1071 noo 0 0 0 0
1072 jumpr 27 0 0 0
1073 noo 0 0 0 0
1074 noo 0 0 0 0
1075 load 21 0 0 247
1076 load 22 0 0 271
1077 setlt 21 22 23 0
1078 setlt 22 21 24 0
1079 or 23 24 23 0
1080 pbranch 23 0 0 0
1081 branchz 0 0 0 8
1082 load 21 0 0 271
1083 store 21 0 0 261
1084 store 21 0 0 260
1085 loadi 27 0 0 973
1086 jump 0 0 0 941
1087 noo 0 0 0 0
1088 noo 0 0 0 0
1089 noo 0 0 0 0
1090 jump 0 0 0 256
1091 noo 0 0 0 0
1092 load 22 0 0 263
1093 setlt 0 22 23 0
1094 not 23 0 23 0
1095 pbranch 23 0 0 0
1096 branchz 0 0 0 1
1097 jumpr 28 0 0 0
1098 jump 0 0 0 973
1099 noo 0 0 0 0
1100 noo 0 0 0 0
1101 load 21 0 0 260
1102 store 21 0 0 228
1103 store 0 0 0 229
1104 store 0 0 0 230
1105 store 0 0 0 231
1106 store 0 0 0 232
1107 loadi 27 0 0 1365
1108 jump 0 0 0 461
1109 noo 0 0 0 0
1110 noo 0 0 0 0
1111 noo 0 0 0 0
1112 load 21 0 0 260
1113 store 21 0 0 280
1114 loadi 27 0 0 1371
1115 jump 0 0 0 1306
1116 jump 0 0 0 257
1117 noo 0 0 0 0
1118 noo 0 0 0 0
1119 noo 0 0 0 0
1120 input 21 0 0 0
1121 store 21 0 0 283
1122 load 21 0 0 234
1123 store 21 0 0 282
1124 loadi 27 0 0 1404
1125 jump 0 0 0 1381
1126 loadi 0 0 0 0
1127 loadi_hd 21 1 0 0
1128 store 21 0 0 284
```

```
1129 load 21 0 0 284
1130 load_hd 21 0 22 0
1131 setlt 0 22 23 0
1132 pbranch 23 0 0 0
1133 branchz 0 0 0 12
1134 load 23 0 0 282
1135 setlt 22 23 24 0
1136 setlt 23 22 23 0
1137 or 23 24 23 0
1138 not 23 0 23 0
1139 pbranch 23 0 0 0
1140 branchz 0 0 0 3
1141 load 23 0 0 283
1142 store_hd 21 0 23 0
1143 jump 0 0 0 1401
1144 addi 21 0 21 32
1145 jump 0 0 0 1383
1146 noo 0 0 0 0
1147 noo 0 0 0 0
1148 jumpr 27 0 0 0
1149 noo 0 0 0 0
1150 jump 0 0 0 299
1151 noo 0 0 0 0
1152 noo 0 0 0 0
1153 load 21 0 0 234
1154 store 21 0 0 285
1155 loadi 27 0 0 1427
1156 jump 0 0 0 1412
1157 load 23 0 0 285
1158 loadi_hd 21 1 0 0
1159 load_hd 21 0 22 0
1160 setlt 22 23 24 0
1161 setlt 23 22 25 0
1162 or 24 25 24 0
1163 pbranch 24 0 0 0
1164 branchz 0 0 0 2
1165 addi 21 0 21 32
1166 jump 0 0 0 1414
1167 loadi 22 0 0 1
1168 store_hd 21 0 22 0
1169 noo 0 0 0 0
1170 noo 0 0 0 0
1171 loadr 27 0 0 0 0
1172 noo 0 0 0 0
1173 noo 0 0 0 0
1174 jump 0 0 0 299
1175 noo 0 0 0 0
1176 noo 0 0 0 0
1177 input 21 0 0 0
1178 store 21 0 0 286
1179 loadi 27 0 0 1458
1180 jump 0 0 0 1436
1181 loadi_hd 21 1 0 0
1182 load 23 0 0 286
1183 load_hd 21 0 22 0
1184 setlt 0 22 24 0
1185 pbranch 24 0 0 0
1186 branchz 0 0 0 11
1187 setlt 22 23 24 0
1188 setlt 23 22 25 0
```

```
1189 or 24 25 24 0
1190 not 24 0 24 0
1191 pbranch 24 0 0 0
1192 branchz 0 0 0 3
1193 loadi 23 0 0 1
1194 store 23 0 0 287
1195 jumpr 27 0 0 0
1196 addi 21 0 21 32
1197 jump 0 0 0 1438
1198 store 0 0 0 287
1199 noo 0 0 0 0
1200 noo 0 0 0 0
1201 jumpr 27 0 0 0
1202 noo 0 0 0 0
1203 noo 0 0 0 0
1204 load 21 0 0 287
1205 setlt 0 21 22 0
1206 pbranch 22 0 0 0
1207 branchz 0 0 0 1
1208 jump 0 0 0 299
1209 noo 0 0 0 0
1210 noo 0 0 0 0
1211 loadi 27 0 0 1486
1212 jump 0 0 0 1468
1213 loadi_hd 21 1 0 0
1214 store 0 0 0 289
1215 load_hd 21 0 22 0
1216 setlt 0 22 23 0
1217 pbranch 23 0 0 0
1218 branchz 0 0 0 9
1219 loadi 23 0 0 1
1220 setlt 23 22 23 0
1221 pbranch 23 0 0 0
1222 branchz 0 0 0 5
1223 addi 21 0 21 32
1224 load 22 0 0 289
1225 addi 22 0 22 1
1226 store 22 0 0 289
1227 jump 0 0 0 1470
1228 noo 0 0 0 0
1229 noo 0 0 0 0
1230 jumpr 27 0 0 0
1231 noo 0 0 0 0
1232 noo 0 0 0 0
1233 load 23 0 0 286
1234 load 22 0 0 289
1235 loadi_hd 21 1 0 0
1236 loadi 24 0 0 32
1237 times 22 24 24 0
1238 add 21 24 21 0
1239 store_hd 21 0 23 0
1240 addi 21 0 21 5
1241 store_hd 21 0 22 0
1242 noo 0 0 0 0
1243 noo 0 0 0 0
1244 jump 0 0 0 299
1245 noo 0 0 0 0
1246 noo 0 0 0 0
1247 load 21 0 0 234
1248 store 21 0 0 292
```

```
1249 loadi 27 0 0 1575
1250 jump 0 0 0 1506
1251 input 21 0 0 0
1252 store 21 0 0 293
1253 loadi 21 0 0 68175
1254 output 21 0 0 0
1255 input 21 0 0 0
1256 store 21 0 0 294
1257 loadi 21 0 0 58175
1258 output 21 0 0 0
1259 input 21 0 0 0
1260 store 21 0 0 295
1261 loadi 21 0 0 58175
1262 output 21 0 0 0
1263 input 21 0 0 0
1264 store 21 0 0 296
1265 loadi 21 0 0 58175
1266 output 21 0 0 0
1267 input 21 0 0 0
1268 store 21 0 0 297
1269 loadi 21 0 0 108175
1270 output 21 0 0 0
1271 input 21 0 0 0
1272 store 21 0 0 298
1273 load 22 0 0 294
1274 shiftl 22 0 22 15
1275 shiftl 22 0 22 11
1276 store 22 0 0 294
1277 load 22 0 0 295
1278 shiftl 22 0 22 15
1279 shiftl 22 0 22 6
1280 store 22 0 0 295
1281 load 22 0 0 296
1282 shiftl 22 0 22 15
1283 shiftl 22 0 22 1
1284 store 22 0 0 296
1285 load 22 0 0 297
1286 shiftl 22 0 22 11
1287 store 22 0 0 297
1288 load 21 0 0 298
1289 add 21 22 21 0
1290 load 22 0 0 296
1291 add 21 22 21 0
1292 load 22 0 0 295
1293 add 21 22 21 0
1294 load 22 0 0 294
1295 add 21 22 21 0
1296 store 21 0 0 299
1297 load 23 0 0 292
1298 loadi_hd 21 1 0 0
1299 load_hd 21 0 22 0
1300 setlt 22 23 24 0
1301 setlt 23 22 25 0
1302 or 24 25 24 0
1303 pbranch 24 0 0 0
1304 branchz 0 0 0 2
1305 addi 21 0 21 32
1306 jump 0 0 0 1554
1307 addi 21 0 21 5
1308 load_hd 21 0 22 0
```

```
loadi_hd 21 2 0 0
loadi 23 0 0 192
times 22 23 22 0
add 21 22 21 0
load 22 0 0 293
add 21 22 21 0
load 23 0 0 299
store_hd 21 0 23 0
noo 0 0 0 0
noo 0 0 0 0
jumpr 27 0 0 0
noo 0 0 0 0
noo 0 0 0 0
jump 0 0 0 299
halt 0 0 0 0
add 0 0 0 0
```

B.2 Mapa do Código Fonte em Pseudocódigo

```
---
menu 0: input: (menu_0_option)
  compare: (menu_0_option) == 0
    ---
    ---
    ---
    ---
    branch: where_am_i_0 [line 28]
  compare: (menu_0_option) == 1
      ---
      ---
      ---
      ---
      branch: list_running_processes [line 31]
    compare: (menu_0_option) == 2
      ---
      ---
      ---
      ---
      branch: command_prompt [line 44]
    compare: (menu_0_option) == 3
      ---
      ---
      ---
      ---
      branch: kill_process [line 37?]
     jump: select_running_process [line 40?]
function: where_am_i_0
  ---
  jump: menu_0 [line 1]
function: list_running_processess
  compare: (running_processes[i]==0)
    ---
    ---
    ---
    ---
    ---
    ---
    ---
    true: jump menu_0 [line 1]
```

```
    false: output running_processess[i]
      i = i + 1;
      jump: compare: (running_processes[i]==0) [line 33]
function: command_prompt
  --menu 1: input: (menu_1_option)
    --compare: (menu_1_option) == 0
      ---
      ---
      ---
      branch: where_am_i_1 [line 64]
    compare: (menu_1_option) == 1
      ---
      ---
      ---
      ---
      branch: list_files [line 68]
    compare: (menu_1_option) == 2
      ---
      ---
      ---
      ---
      branch: previous_menu_1 [line 67]
      jump: compare: (menu_1_option) == 3 [line 88]
function: where_am_i_1
  ---
  jump: menu_1 [line 44]
function: previous_menu_1 jump: menu_0 [line 1]
function: list_files
  compare: (files[i])==0
  ---
  ---
  ---
  ---
  ---
  ---
  ---
  true: jump menu_1 [line 44]
  false: compare: (files[i]) == 1
    ---
    ---
    ---
    ---
    ---
    false: output files[i]
    true:
      i = i+1
      jump list_files [line 70]
      compare: (menu_1_option) == 3
        ---
        ---
        ---
        ---
        true: branch: create_file [line 1177]
false: function: select_file
---  routine: verify_if_running
    compare: (menu_1_option) == 0
      ---
      ---
      ---
      ---
```

```
      true: branch [line 111]
      false: compare: (menu_1_option) == running_processes[i]
        ---
        ---
        ---
        ---
        false: i = i+1; jump: verify_if_running [line 96]
        true: jump menu_1 [line 44]
        ---
        ---
    routine: verify_if_exists
    compare: hd[1][i] == 0
      ---
      ---
      ---
      ---
      ---
      true: jump menu_1 [line 44]
      ---
      false: compare: hd[1][i] == menu_1_option
        ---
        ---
        ---
        true: branch [line 127]
        false: i = i + 1; jump: verify_if_exists [line 111]
          ---
  variable: file_selected memory[0?]
  input: (select_file_menu_option)
    compare: (select_file_menu_option) == 0
      ---
      ---
      ---
      ---
      ---
      true: function: where_am_i_file
        variable (file_selected memory)
        ---
        jump: select_file_menu_option [line 128]
      false: compare: (select_file_menu_option) == 2
        ---
        ---
        ---
        ---
        ---
        ---
        true: jump: menu_1 [line 44]
        false: jump verify_if_1 [line 351]
load return [line 582]
---
  function is_list_running_available;
  place = 0; memory_position = 0;
  i = 0;
    while (list_running_processes[i]!=0) {//not end of list
      ---
      ---
      ---
      ---
      ---
      ---
      ---
```

```
      if (list_running_processes[i]!=1) {//not empty space
        ---
        ---
        ---
        ---
        ---
        ---
        if (list_running_processes[i].place==place) {//if it is in memory
          ---
          ---
          ---
          ---
          ---
          ---
          ---
          ---
            memory_position++;
            ---
            }
          }
          i++;
          (back to first while)}
    if (memory_position>2) {
      ---
      ---
      ---
      ---
      place = 1;
        ---
        }
      return place;
      ---
      jump:  run_process [line 195]
      ends: 240
---
---
load return [line 241]
update_program_info.id = selected_file
---
update_program_info.index_process = 0
update_program_info.place = 0
update_program_info.state = 1
---
update_program_info.program_counter = 0
call update_program_info;
function update_program_info
  i = 0;
  ---
        while(list_programs_info[i]!=0){
      ---
      ---
      ---
      ---
      ---
      ---
      ---
                if(list_programs_info[i].id == id){
            ---
            ---
            ---
```

```
            ---
            ---
            ---
            list_programs_info[i].index_process = index_process;
            list_programs_info[i].place = place;
            list_programs_info[i].state = state;
            list_programs_info[i].program_counter = program_counter;
            break;
            ---
            ---
            ---
            ---
            ---
            ---
            ---
            }
          i++;
          }
          ---
          return from update_program_info
load return [line 271]
insert_list_running.id = selected_file
---
insert_list_running.place = 0//memory
call insert_list_running
function: insert_list_running
  i = 0
  while(list_running[i].id!=0){
    ---
    ---
    ---
    ---
    ---
                if(list.running[i].id==1){
      ---
      ---
      ---
      break
    }
    i++;
    ---
    ---
    ---
  list_running[i].id = id;
  ---
  ---
  list_running[i].place = place;
  ---
  ---
  return from insert_list_running
get_index_hd.id = selected_file
---
load return [line 300]
function get_index_hd(id){
        i = 0;
        while(list_programs_info[i]!=0){
                if(list_programs_info[i].id == id){
                        index_hd = list_programs_info.index_hd;
                        break;
                }
```

```
281                 i++;
282         }
283         return index_hd;
284   ---
285   ---
286   ---
287   ---
288   ---
289   ---
290   ---
291   ---
292   ---
293   ---
294   ---
295   ---
296   ---
297   ---
298   ---
299   ---
300 }
301 transfer_hd_to_iram.index_hd = get_index_hd.index_hd
302 ---
303 call transfer_hd_to_iram
304 function transfer_hd_to_iram(index_hd){
305         i = 0;
306         index = index_hd*64;
307         while(list_programs[index+i] !=0){
308                 instructions_memory[i] = list_programs[index+i];
309                 i++;
310         }
311   ---
312   ---
313   ---
314   ---
315   ---
316   ---
317   ---
318   ---
319   ---
320   ---
321   ---
322   ---
323   ---
324   return from transfer_hd_to_iram
325 }
326 PC = 0
327 load return [line 343]
328 call is_list_running_empty
329 function is_list_running_empty(){
330         i = 0;
331         while(list_running[i] !=0){
332                 if(list_running[i]!=1) i=1;
333         }
334         return i;
335   ---
336   ---
337   ---
338   ---
339   ---
340   ---
```

```
341   return from is_list_running_empty
342 }
343 if(is_list_running_empty==0)
344   jump [line 196]
345 else
346   jump [line 377]
347   ---
348   ---
349   ---
350   ---
351 compare: (select_file_menu_option) == 1
352   true: running_empty [line 327]
353   false:
354   ---
355   ---
356   ---
357   ---
358   ---
359   compare: (select_file_menu_option) == 3
360     true: call rename_file [line ?]
361     false:
362     ---
363     ---
364     ---
365     compare: (select_file_menu_option) == 4
366       true: call delete_file [line ?]
367       false:
368       ---
369       ---
370       ---
371       compare: (select_file_menu_option) == 5
372         true: call edit_file [line ?]
373         false:
374         ---
375         ---
376         ---
377   load return [line 379]
378   call is_list_running_available
379   compare: (is_list_running_available) == 0
380   false: update_index_process_HD
381   true:
382   ---
383   ---
384   ---
385   ---
386   ---
387   load return [line 423]
388   function get_running_process{
389     i = 0;
390     running_process = 0;
391     while(list_programs_info[i]!=0){
392       if(list_programs_info[i]!=1); info.state == 1
393       if(list_programs_info[i].index_process==0){
394         running_process = list_programs_info[i].id;
395         break;
396       }
397       i++;
398     }
399     ---
400     ---
```

```
401     ---
402     ---
403     ---
404     ---
405     ---
406     ---
407     ---
408     ---
409     ---
410     ---
411     ---
412     ---
413     ---
414     ---
415     ---
416     ---
417     ---
418     ---
419     ---
420     ---
421     return running_process;
422   }
423   load return [line 478]
424   call get_memory_position_available
425   function get_memory_position_available(){
426     i = 0;
427     position_available = 3;
428     array_memory_positions[3] = 0;
429       while (list_programs_info[i].id!=0) {
430         if (list_programs_info[i].id!=1) {
431           if (list_programs_info[i].state==1) {
432             array_memory_positions[list_programs_info[i].index_process] = 1;
433           }
434         }
435         i++;
436     }
437
438     if (array_memory_positions[1]==0) {
439       position_available = 1;
440     }if(array_memory_positions[2]==0){
441       position_available = 2;
442     }
443
444     return position_available;
445     ---
446     ---
447     ---
448     ---
449     ---
450     ---
451     ---
452     ---
453     ---
454     ---
455     ---
456     ---
457     ---
458     ---
459     ---
460     ---
```

```
461     ---
462     ---
463     ---
464     ---
465     ---
466     ---
467     ---
468     ---
469     ---
470     ---
471     ---
472     ---
473     ---
474     ---
475     ---
476     ---
477     ---
478     ---
479     ---
480     ---
481   }
482   update_memory_position.id = get_running_process.running_process
483   ---
484   update_memory_position.new_index_process = get_memory_position_available.
          position_available
485   ---
486   load return [line 529]
487   call update_memory_position
488   function update_memory_position(id, new_index_process){
489   i = 0;
490   former_index_process = 0;
491   new_index_process_aux;
492   while (list_programs_info[i]!=0) {
493     if(list_programs_info[i].id==id){
494       former_index_process = list_programs_info[i].index_process;
495       list_programs_info[i].index_process = new_index_process;
496       break;
497     }
498     i++;
499   }
500
501   i=0;
502   former_index_process = former_index_process*size_of_section;
503   new_index_process_aux = new_index_process*size_of_section;
504   while (i<size_of_section) {
505     running_processes[new_index_process_aux+i] = running_processes[former_index_process+i
          ];
506     i++;
507   }
508   ---
509   ---
510   ---
511   ---
512   ---
513   ---
514   ---
515   ---
516   ---
517   ---
518   ---
```

```
---
---
---
---
---
---
---
---
---
---
---
---
}
save_snapshot.id = get_running_process.id;
--
load return [line 198]
call save_snapshot
function save_snapshot(running_process){
i = 0;
while (list_programs_info[i]!=0) {
  if (list_programs_info[i].id==running_process) {
    break;
  }
  i++;
}
i = i+12;
HD[i] = R[0];
i++;
HD[i] = R[1];
... HD[i] = R[19]
---
---
---
---
---
---
---
---
---
---
---
---
---
---
---
---
---
---
---
---
---
---
---
---
---
---
---
---
---
---
```

```
  ---
  ---
  ---
  ---
  ---
  ---
  ---
  ---
  ---
  ---
  ---
  ---
  ---
  ---
  ---
  ---
  ---
  ---
  ---
  }
if(menu_0_option<10){
  call something
}else{
---
---
selected_process = menu_0_option;
check_if_running.id = menu_0_option;//257
load return [line 635]
call check_if_running
function check_if_running(id){
  i = 0;
  is_running = 0;
  while (list_running[i]!=0) {
    if(list_running[i].id == id){
      is_running = 1;
      break;
    }
  }
  return is_running;
  ---
  ---
  ---
  ---
  ---
  ---
  ---
  ---
  ---
  ---
  ---
  ---
  ---
  ---
  ---
  ---
}
if(check_if_running.is_running==0){
  call menu_0;
}else{//menu_access_process
  input(access_process_menu_option);
```

```
639   if (access_process_menu_option==0) {
640     output(selected_process);
641     call menu_access_process; [line 641]
642   }else if (access_process_menu_option==2) {//not 22 0 22 0
643     call menu_0; [line 257]
644   }else if(access_process_menu_option==3){
645     call kill_process;//input 21
646   }
647   else if(3<access_process_menu_option){
648     call menu_access_process; [line 641]
649   }
650   ---
651   ---
652   ---
653   ---
654   ---
655   ---
656   ---
657   ---
658   ---
659   ---
660   ---
661   ---
662   ---
663   ---
664   ---
665   ---
666   ---
667   ---
668   ---
669   ---
670   ---
671   ---
672   ---
673   ---
674   ---
675   ---
676   ---
677   ---
678   ---
679   ---
680   else if(access_process_menu_option==1){
681     get_index_process.id = selected_process;
682     ---
683     load return [line 1092];
684     call get_index_process;/protagonist_program = selected_process
685 function get_index_process{
686   i = 0;
687   index_process;
688     while (list_programs_info[i]!=0) {
689       if (list_programs_info[i]!=1) {
690         if (list_programs_info[i].id == id) {
691           index_process = list_programs_info.index_process;
692           break;
693         }
694       }
695       i++;
696     }
697     return index_process;
698     ---
```

```
    ---
    ---
    ---
    ---
    ---
    ---
    ---
    ---
    ---
    ---
    ---
    ---
    ---
    ---
    ---
    ---
}
    update_memory_position.id = selected_process;
    ---
    update_memory_position.new_index_process = 4;
    ---
    load return [line 724]
    call update_memory_position;
    load return [line 726]
    call get_running_process;
    update_memory_position.id = get_running_process.id;
    ---
    update_memory_position.new_index_process = get_index_process.index_process;
    ---
    load return [line 732]
    call update_memory_position;
    update_memory_position.id = selected_process;
    ---
    update_memory_position.new_index_process = 0;
    load return [line 737]
    call update_memory_position;
    save_snapshot.id = get_running_process.id;
    ---
    load return [line 741]
    call save_snapshot;
    resume_snapshot.id = selected_process.id;
    ---
    load return [line 812]
    call resume_snapshot;
function resume_snapshot(id){
  i = 0;
  while (list_programs_info[i]!=0) {
    if (list_programs_info[i].id==id) {
      break;
    }
    i++;
  }
  i = i+12;
  R[0] = HD[i];
  i++;
  R[1] = HD[i];
  ...   R[19] = HD[i];
  ---
```

```
759   ---
760   ---
761   ---
762   ---
763   ---
764   ---
765   ---
766   ---
767   ---
768   ---
769   ---
770   ---
771   ---
772   ---
773   ---
774   ---
775   ---
776   ---
777   ---
778   ---
779   ---
780   ---
781   ---
782   ---
783   ---
784   ---
785   ---
786   ---
787   ---
788   ---
789   ---
790   ---
791   ---
792   ---
793   ---
794   ---
795   ---
796   ---
797   ---
798   ---
799   ---
800   ---
801   ---
802   ---
803   ---
804   ---
805   ---
806   ---
807   ---
808   ---
809   ---
810   ---
811     }
812     load return [line ?]
813     get_index_hd.id = selected_process.id;
814     ---
815     call get_index_hd(id);
816     transfer_hd_to_iram.index_hd = get_index_hd.index_hd;
817     ---
818     load return [line ?]
```

```
819     call transfer_hd_to_iram
820     PC <= R[28]
821   }
822 }
823 ---
824 ---
825 ---
826 ---
827 ---
828 load return;
829 call context_exchange_selection(output_watchdog);
830 function context_exchange_selection(output_watchdog){
831   selector = output_watchdog;
832   if (selector>0) {
833     if (selector<4) {
834       call context_exchange(selector); [line 853]
835     }else if (selector<5) {
836       call treat_waiting(); [line 955]
837     }else if (selector<6) {
838       call treat_halt();
839     }
840     ---
841     ---
842     ---
843     ---
844     ---
845     ---
846     ---
847     ---
848     ---
849     ---
850     ---
851   }
852 }
853 ---
854 ---
855 ---
856 ---
857 ---
858 load return [line 900]
859 get_id_from_index_process.index_process = context_exchange_selection.selector;
860 ---
861 call get_id_from_index_process/90;
862 function get_id_from_index_process(index_process){
863   i = 0;
864   id = 0;
865   while (list_programs_info[i]!=0) {
866     if (list_programs_info[i]!=1) {
867       if (list_programs_info[i].state>0) {
868         if (list_programs_info[i].state<3) {
869           if (list_programs_info[i].index_process==index_process) {
870             id = list_programs_info[i].id;
871           }
872         }
873       }
874     }
875     i++;
876   }
877   return id;
878   ---
```

```
  ---
  ---
  ---
  ---
  ---
  ---
  ---
  ---
  ---
  ---
  ---
  ---
  ---
  ---
  ---
  ---
  ---
  ---
}
if(id==0)
  return execution: jump r[28];
  ---
  ---
  ---
  ---
  ---
  else{
    ---
    ---
    ---
    get_state.id = get_id_from_index_process.id;
    ---
    load return [line 939]
    call get_state();
    function get_state(id){
      id = id;
      state = 0;
      i = 0;
      while (list_programs_info[i]!=0) {
        if (list_programs_info[i].id==id) {
          state = list_programs_info[i].state;
        }
        i++;
      }
      return state;
      ---
      ---
      ---
      ---
      ---
      ---
      ---
      ---
      ---
      ---
      ---
      ---
    }
```

```
939     if(get_state.state==2){
940       return execution: jump r[28];
941       ---
942       ---
943       ---
944       ---
945       ---
946       ---
947       ---
948       ---
949       ---
950     }else{
951       get_index_process.id = get_id_from_index_process.id;
952       load return [line 718]
953       call get_index_process();
954     }
955     ---
956     ---
957     ---
958   }
959   call get_running_process(); [line 388]
960   ---
961   if (get_running_process.running_process==protagonist_program) {
962     return_to_execution; [line r[28]]
963     ---
964     ---
965     ---
966     ---
967     ---
968     ---
969     ---
970     ---
971   }else{
972     ---
973     ---
974     load return 983;
975     update_program_info.id = get_running_process.running_process;
976     ---
977     update_program_info.index_process = 0;
978     update_program_info.place = 0;
979     update_program_info.state = 2;
980     ---
981     update_program_info.program_counter = r[28];
982     call update_program_info; [line 206]
983     ---
984     ---
985     get_index_process.id = protagonist_program;
986     ---
987     get_index_process.id = protagonist_program;
988     selected_process = protagonist_program;
989     call get_index_process; //change context
990     ---
991     ---
992     ---
993   }
994   menu_0_option = menu_0_option (r[21]);
995   if (menu_0_option<4) {
996     ---
997     ---
998     ---
```

```
999     load return [line 1022];
1000    call show_waiting_processes(); [line 1001]
1001    function show_waiting_processes(){
1002      i = 0;
1003      while (list_programs_info[i]!=0) {
1004        if (list_programs_info[i].state>=2) {
1005          output(list_programs_info[i].id);
1006        }
1007        i++;
1008      }
1009      ---
1010      ---
1011      ---
1012      ---
1013      ---
1014      ---
1015      ---
1016      ---
1017      ---
1018      ---
1019      ---
1020      ---
1021    }
1022    menu_0_option = 0;
1023    jump menu_0 [line 0];
1024  }else{
1025    jump somewhere;
1026    ---
1027    ---
1028  }
1029  ---
1030  ---
1031  ---
1032  ---
1033 load return [line 1035];
1034 call get_running_process(); [line 388]
1035 update_program_info.id = get_running_process.running_process;
1036 ---
1037 update_program_info.index_process = 0;
1038 update_program_info.place = 0;
1039 update_program_info.state = 0;
1040 update_program_info.program_counter = 0;
1041 load return [line 1043];
1042 call update_program_info(); [line 206]
1043 ---
1044 ---
1045 ---
1046 ---
1047 load return [line 1073];
1048 remove_from_list_running.id = get_running_process.running_process;
1049 ---
1050 call remove_from_list_running();
1051 function remove_from_list_running(id){
1052   i = 0;
1053   while (list_running!=0) {
1054     if (list_running[i].id==id) {
1055       list_running[i] = 1;
1056     }
1057     i++;
1058   }
```

```
1059    ---
1060    ---
1061    ---
1062    ---
1063    ---
1064    ---
1065    ---
1066    ---
1067    ---
1068    ---
1069    ---
1070    ---
1071    ---
1072  }
1073  ---
1074  ---
1075  if(get_running_process.id!=protagonist_program){
1076    ---
1077    ---
1078    ---
1079    ---
1080    ---
1081    ---
1082    get_index_process.id = protagonist_program;
1083    ---
1084    selected_process = protagonist_program;
1085    load return [line 990]
1086    call get_index_process();
1087    ---
1088    ---
1089    ---
1090    ---
1091  }else{jump [line 1100]}
1092  if(selected_process==running_program){
1093    jumpr r[28];//  return to execution;
1094  }else{
1095    jump [line 973];//procceed to change context
1096    ---
1097    ---
1098  }
1099  ---
1100  ---
1101  update_program_info.id = selected_process;
1102  ---
1103  update_program_info.index_process = 0;
1104  update_program_info.place = 0;
1105  update_program_info.state = 0;
1106  update_program_info.program_counter = 0;
1107  load return [line 1110];
1108  call update_program_info(); [line 206]
1109  ---
1110  ---
1111  ---
1112  remove_from_list_running.id = selected_process;
1113  ---
1114  load return [line 1116];
1115  call remove_from_list_running();
1116  return menu_0 [line 0];
1117  ---
1118  ---
```

```
---
input(new_id);
update_program_id.new_id = new_id;
update_program_id.previous_id = selected_file;
---

load return [line 1149];
call update_program_id;
update_program_id(previous_id, new_id){
int i;
i = 0;
while (list_programs_info[i].id!=0) {
  if (list_programs_info[i].id==previous_id) {
    list_programs_info[i].id = new_id;
    break;
  }
}
---
---
---
---
---
---
---
---
---
---
---
---
}
---
return (menu_1) [line 44]
---
---
delete_file.id = selected_file;
---
load return [line 1172];
call delete_file();
function delete_file(id){
  i = 0;
  while (list_programs_info[i]!=id) {
    i = i + 1;
  }
  list_programs_info[i] = 1;
  ---
  ---
  ---
  ---
  ---
  ---
  ---
  ---
}
---
---
return menu_1 [line 44];
---
---
input(id);
check_if_exists_list_programs_info.id = id;
```

```
load return [line 1203];
call check_if_exists_list_programs_info();
function check_if_exists_list_programs_info(id){
  i = 0;
  while (list_programs_info[i]!=0) {
    if (list_programs_info[i]==id) {
      return 1;
    }
    i = i + 1
  }
  return 0;
  ---
  ---
  ---
  ---
  ---
  ---
  ---
  ---
  ---
  ---
  ---
}
---
---
if(check_if_exists_list_programs_info.exists == 1){
  return menu_1 [line 44];
  ---
  ---
  ---
}else{
---
---
load return [line 1231];
call get_hd_position_available();
function get_hd_position_available(){
  i = 0;
  position_hd = 0;

  while (list_programs_info[i].id!=0) {
    if (list_programs_info[i].id==1) {
      break;
    }
    i = i + 1;
  }
  position_hd = i;
  ---
  ---
  ---
  ---
  ---
}
---
---
list_programs_info[32*position_available+start] = check_if_exists_list_programs_info.id;
list_programs_info[32*position_available+start+5] = get_index_hd.position_available;
---
---
---
---
```

```
1239 ---
1240 ---
1241 ---
1242 ---
1243 ---
1244 return menu_1 [line 44];
1245
1246
1247 }
```