Projeto de Sistemas Operacionais - Projeto 1.1

Jhonattan C. B. Cabral[1] **Igor Barbosa Nogueira**[1]

[1]Departamento de Informática e Matemática Aplicada (DIMAp)
Universidade Federal do Rio Grande do Norte (UFRN)

jhonattan.yoru@gmail.com, igornogueir@gmail.com

1. Introdução

Para compor uma das notas da primeira unidade das disciplinas de Projetos Operacionais [DIM0615.0] e Laboratório de Projetos Operacionais [DIM0615.1] tivemos como objetivo a implementação de um projeto onde seria necessário verificar o consumo total de CPU e memória e representar a porcentagem total acendendo alguns LEDs.

2. Desenvolvimento

2.1. Funções

2.1.1. menuSystem();

```
1  void menuSystem()
2  {
3      int c = 0;
4
5      std::cout << std::endl;
6      std::cout << "\t Monitoramento" << std::endl;
7      std::cout << std::endl;
8      std::cout << "1. Monitorar CPU" << std::endl;
9      std::cout << "2. Monitorar Memoria" << std::endl;
10     std::cout << "0. Sair" << std::endl;
11
12     std::cin >> c;
13
14     switch (c)
15     {
16     case 1:
17         monitorar("CPU");
18         break;
19
20     case 2:
21         monitorar("MEMORY");
22         break;
23
24     case 0:
25         exit(0);
26         break;
27     }
28 }
```

Menu de controle e usablidade do sistema.

2.1.2. gpioReady();

```
bool gpioReady(int pin, string direction) {

    if (pin < 1) {
        cerr << "Pin Invalido!" << endl;
        return false;
    }

    if (!(direction == "out" || direction == "in")) {
        cerr << "Direcao invalida!" << endl;
        return false;
    }

    string _pin = to_string(pin);

    string comando = "echo " + _pin + " > /sys/class/gpio/export
       ";

    system(comando.c_str());

    comando = "echo " + direction + " > /sys/class/gpio/gpio"+
       _pin+"/direction";

    system(comando.c_str());

    return true;

}
```

Prepara as portas gpio e muda a direção do fluxo de dados delas.

2.1.3. gpioFinish();

```
bool gpioFinish(int pin) {

    if (pin < 1) {
        cerr << "Pin Invalido!" << endl;
        return false;
    }

    string _pin = to_string(pin);

    string comando = "echo " + _pin + " > /sys/class/gpio/
       unexport";

    system(comando.c_str());

```

```
14      return true;
15  }
```

Fecha o acesso a uma determinada porta gpio.

2.1.4. signalHandler();

```
1   void signalHandler(int sig) {
2
3       gpioFinish(LED_VERMELHO);
4       gpioFinish(LED_VERDE);
5       gpioFinish(LED_AMARELO);
6       gpioFinish(BUTTON);
7
8       cout << "Programa finalizado com sucesso!\n" << endl;
9       exit(0);
10  }
```

Função que será executada quando o processor recebe um sinal de kill.

2.1.5. valueLow();

```
1   void valueLow(int pin) {
2       string _pin = to_string(pin);
3
4       string comando = "echo 0 > /sys/class/gpio/gpio" + _pin + "/
            value";
5
6       system(comando.c_str());
7   }
```

Seta valor lógico 0 para a porta GPIO que é passada.

2.1.6. valueHigh();

```
1   void valueHigh(int pin)
2   {
3       string _pin = to_string(pin);
4
5       string comando = "echo 1 > /sys/class/gpio/gpio" + _pin + "/
            value";
6
7       system(comando.c_str());
8   }
```

Seta valor lógico 1 para a porta GPIO que é passada.

2.1.7. blinkLed();

```
1   void blinkLed(int led1, int led2, int led3) {
2
```

```
    valueHigh(led1);
    valueHigh(led2);
    valueHigh(led3);
    valueLow(led1);
    valueLow(led2);
    valueLow(led3);
}
```

Função que faz os leds piscarem.

2.1.8. buttonPress();

```
bool buttonPress(int pin) {

    if (pin < 1) {
        return false;
    }

    string _pin = to_string(pin);

    string comando = "cat /sys/class/gpio/gpio" + _pin + "/value
       ";

    string resultado = exec(comando.c_str());

    istringstream reader(resultado);

    int aux;

    reader >> aux;

    if (aux == 1) {
        return true;
    } else {
        return false;
    }
}
```

Verifica se um botão está pressionado ou não. Retorna $TRUE$ caso esteja com valor lógico 1 e $FALSE$ caso esteja com valor lógico 0.

2.1.9. dontPanic();

```
void dontpanic(string pid, int led1, int led2, int led3, int
   button) {

    while (1) {
        blinkLed(led1, led2, led3);
        if (buttonPress(button)) {
            break;
```

```
        }
    }

    killProcess(pid);

    valueLow(LED_VERDE);
    valueLow(LED_VERMELHO);
    valueLow(LED_AMARELO);

    sleep_until(system_clock::now() + seconds(3));


    return;
}
```

Funcao que coloca o programa em estado de panico até o acionamento do botao. Quando o botão é acionado ele tenta matar o processo que estaria consumindo maior quantidade de CPU ou memória.

2.1.10. killProcess();

```
void killProcess(string pid) {

    string comando = "kill -9 " + pid;

    system(comando.c_str());

}
```

Mata o processo do PID passado como argumento.

2.1.11. totalConsumoCPU();

```
float totalConsumoCPU()
{

    string command = "ps h -e -o %cpu --sort=-%cpu";

    string consumoCPU = exec(command.c_str());
    std::istringstream reader(consumoCPU);

    float consumoTotal = 0;
    string line;

    while (reader >> line)
    {
        consumoTotal += atof(line.c_str());
    }
    cout << "Consumo CPU: "<< consumoTotal << endl;
    return consumoTotal;
}
```

Faz o somatorio do consumo total e retorna esse valor no determinado instante.

2.1.12. totalConsumoMEMO();

```
float totalConsumoMEMO()
{

    string command = "ps h -e -o %mem --sort=-%mem";

    string consumoMEMO = exec(command.c_str());
    std::istringstream reader(consumoMEMO);

    float consumoTotal = 0;
    string line;

    while (getline(reader, line))
    {
        consumoTotal += atof(line.c_str());
    }

    cout << "Consumo MEM: " << consumoTotal << endl;

    return consumoTotal;
}
```

Faz o somatorio do consumo total e retorna esse valor no determinado instante.

2.1.13. maiorConsumidor();

```
string maiorConsumidor(bool cpuMemo)
{
    string comando, aux;
    if (cpuMemo) {
        comando = "ps h -e -o pid --sort=-%cpu";
        aux = exec(comando.c_str());
        std::istringstream reader(aux);
        reader >> aux;
        cout << aux;
        return aux;
    } else {
        comando = "ps h -e -o pid --sort=-%mem";
        aux = exec(comando.c_str());
        std::istringstream reader(aux);
        reader >> aux;
        cout << aux;
        return aux;
    }
}
```

Retorna o PID do maior consumidor de CPU/memoria. Quando passa o valor $TRUE$ retorna o PID do maior consumidor de CPU, quando passa $FALSE$ retorna o

PID do maior consumidor de memória.

2.1.14. monitorar();

```
void monitorar(string tipo)
{
    std::cout << std::endl;
    std::cout << "\t Monitoramento "<< tipo << " - Use CTRL C
       para sair" << std::endl;
    std::cout << ">> Em caso de consumo excessivo utilize o
       botao do panico." << std::endl;
    std::cout << std::endl;

    float consumo = 0;
    string pid;
    while (1)
    {
        if (tipo.compare("CPU") == 0) {
            consumo = totalConsumoCPU();
        } else if (tipo.compare("MEMORY") == 0) {
            consumo = totalConsumoMEMO();
        }

        if (consumo > 0 && consumo <= 25)
        {
            valueHigh(LED_VERDE);
            valueLow(LED_AMARELO);
            valueLow(LED_VERMELHO);
        }
        else if (consumo > 25 && consumo <= 50)
        {
            valueLow(LED_VERDE);
            valueHigh(LED_AMARELO);
            valueLow(LED_VERMELHO);
        }
        else if (consumo > 50 && consumo <= 75)
        {
            valueLow(LED_VERDE);
            valueLow(LED_AMARELO);
            valueHigh(LED_VERMELHO);
        }
        else
        { // pegar processo que esta consumindo mais de 75% da
           cpu e passa pro dontpanic junto com os leds
            if (tipo.compare("CPU") == 0) {
                pid = maiorConsumidor(true);
            } else if (tipo.compare("MEMORY") == 0) {
                pid = maiorConsumidor(false);
            }
            dontpanic(pid, LED_VERDE, LED_AMARELO, LED_VERMELHO,
```

```
                    BUTTON);
44          }
45      }
46  }
```

Funcao de monitoramento da CPU ou da memória do computador de acordo com os parâmetros passados.

2.1.15. exec();

```
1   string exec(const char *cmd)
2   {
3       FILE *pipe = popen(cmd, "r");
4       if (!pipe)
5           return "ERROR";
6       char buffer[128];
7       std::string result = "";
8       while (!feof(pipe))
9       {
10          if (fgets(buffer, 128, pipe) != NULL)
11              result += buffer;
12      }
13      pclose(pipe);
14      return result;
15  }
```

Função que executa um comando no terminal e retorna o que seria printao como string. Créditos: https://stackoverflow.com/questions/32039852/returning-output-from-bash-script-to-calling-c-function.

2.2. Circuito

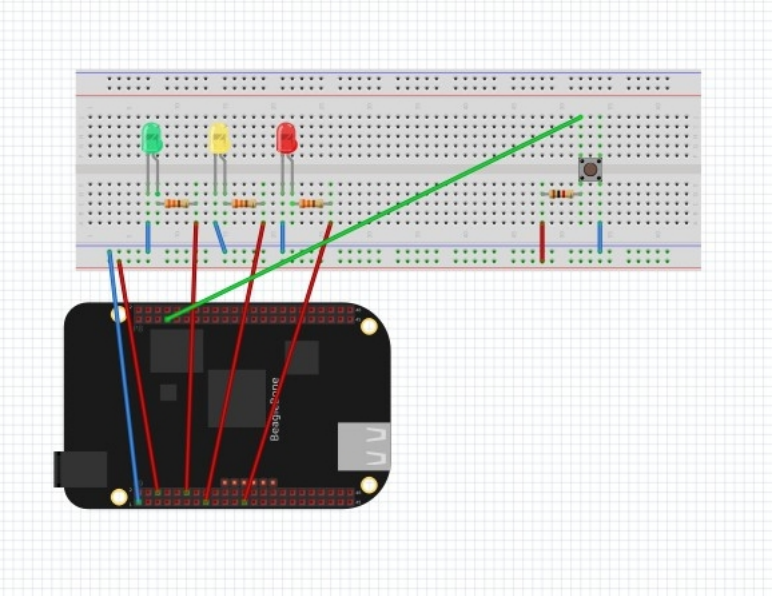

Figura 1. Circuito

O esquema acima mostra o circuito do projeto, feito de maneira simples utilizamos 3 resistores de 330ohm cada para os LEDs, na figura acima mostra um resistor de 1Kohm para o botão de interrupção, porém preferimos depois utilizar a porta Pull Down da Beagle Bone. Como podemos ver utilizamos jumpers para ligar os componentes ao VCC da placa (fio vermelho), para ligar ao GND (fio azul), e utilizamos também um jumper para levar o sinal do botão até a placa (fio verde).

3. Dificuldades

A maior dificuldade foi sincronizar de maneira precisa o consumo da CPU e da memória e assim poder testar a leitura com os LEDs, conseguimos um melhor resultado com a CPU pois utilizando o comando ”cpulimit”a sincronização foi facilitada. Já com a memória, mesmo utilizando o comando ”stress”a Beagle Bone já inicia o sistema consumindo boa parte da memória, com isso tivemos dificuldades em controlar o consumo da memória.

4. Considerações Finais

Embora tendo conseguido um ótimo desempenho apenas na verificação da CPU, tivemos resultados satisfatório já que cumprimos todas as etapas do projeto. Conseguimos verificar bem cada consumo e sincronizar isso com os LED’s. Fizemos um programa extremamente genérico visando a utilização futura de cada componente do código.