Universidade do Minho
Escola de Engenharia

Execução Agendada de Conjuntos de Programas

Mestrado Integrado de Engenharia de Telecomunicações e Informática

Sistemas Operativos

Grupo 5

Francisco Duarte Araújo Rocha

A65232

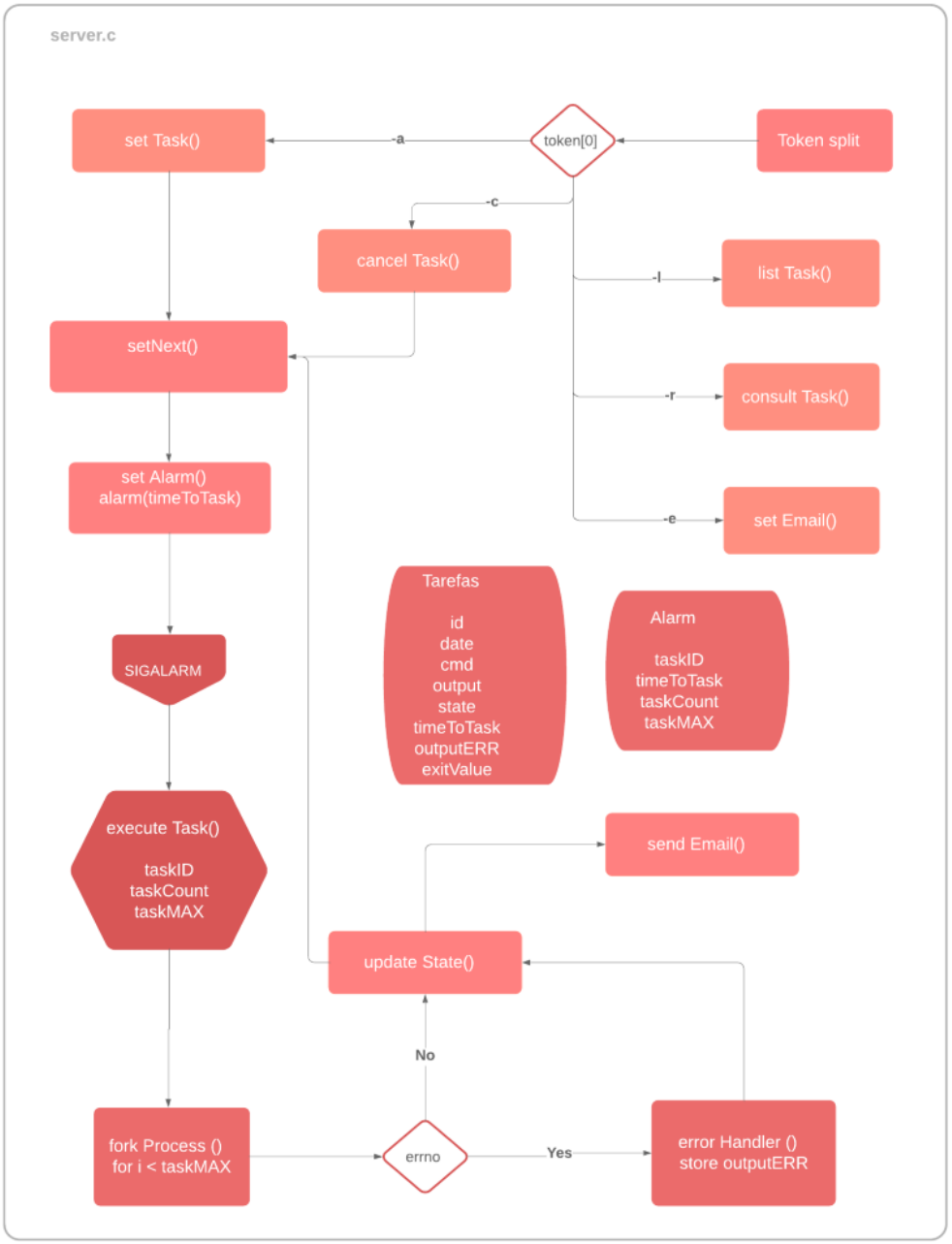
Simão Pedro Tinoco Rocha

A65234

Solução Implementada

Abordagem a este problema passou por ter um cliente que apenas faz um pedido ao servidor e apenas recebe um resultado do pedido efetuado.

O cliente lê do STDIN o comando a executar, escreve no pipe (fechando este de seguida), abre o pipe para leitura e espera pelo resultado.
Quando receber o resultado este é encaminhado para o ecrã (STDOUT).
O servidor, por sua vez, lê do pipe, processa pedido e escreve no pipe o resultado obtido.



set Alarm

Cria um alarm(alarm[0].timeToTask)

- SIGALARM

cancel Task

Recebe o identificador da tarefa e atualiza o seu valor para -2 (valor definido para tarefa cancelada)

set Email

Recebe o email para o qual serão enviados os resultados das tarefas e guarda numa variável.
Este valor poderá ser associado ao alarme caso todas as tarefas sejam enviadas para o mesmo utilizador ou ás tarefas caso seja pretendido enviar e-mails para diferentes utilizadores.

list Task

Recebe um pedido de listagem e escreve no canal de comunicação todas as tarefas, agrupadas por agendadas (*state* = 0), executadas (*state* = 1) e canceladas (*state* = 2).

consult Task

Recebe um pedido consulta e o identificador da tarefa a ser executada.
Escreve no canal de comunicação toda a informação acerca da tarefa pretendida caso esta tenha sido efetuada (*state* = 1).

execute Task

Lê da struct Alarm a seguinte informação:

- Número de tarefas a serem executada;
- Identificador de cada tarefa;
- Número máximo de tarefas a serem executadas concorrentemente.

Verifica se o número de tarefas a executar concorrentemente é maior que o número máximo definido. Caso este seja verdade faz várias iterações.

Para cada tarefa a ser executada concorrentemente (*alarm[0].taskID[]*) são criando n processos que executam concorrentemente cada um dos comandos recebidos *tarefa[n].cmd* (n corresponde ao número máximo de processos a executarem concorrentemente ou ao numero de tarefas para execução, dependendo se este é maior ou menor que o número máximo). É então criado um canal de comunicação entre processos pai e processos filho (PIPE), os descritores de leitura e escrita dos processos filho e pai, respetivamente, são fechados e redirecionados para os descritores de leitura e escrita do canal de comunicação.

O processo pai espera pelo que o seu processo filho termine e guarda em memória o resultado da execução. Em caso de falha de execução o processo Pai faz um *catch* da variável *errno* e guarda em memória o resultado da variável errno(*strerror(WEXITSTATUS(status)*).
Sempre que um processo filho termina é gerado outro processo filho que envia os resultados obtidos para o email pré-definido.

O valor .*state* da tarefa assume o valor de 1 (valor definido para tarefa efetuada), *timeToTask* assume o valor de -1 (valor definido para tarefa efectuada) e todos os resultados guardados em memória são copiados para as variáveis das tarefas correspondentes.

set Next

Precorre todo o array de tarefas e, caso estas não estejam canceladas nem tenham sido exetutadas, calcula a diferença entre a data para qual foi agendada a tarefa e a data do momento em que o servidor recebeu o novo pedido de agendamento e guarda o valor caso este seja menos que o valor na tarefa anterior.
Após calculado tempo para o próximo alarme, percorre novamente todas as tarefas e verifica quais as próximas tarefas a serem executadas, guarando o seu id (alarm(0).taskID[])).
Sempre que uma tarefa é agendada, cancelada ou executada a função set Next é chamada e um novo alarme é calculado

send Email

Para cada processo filho que termine é criado um novo processo filho que envia um email para o utilizador definido em *alarm.emailToSend* ou *tarefa.emailToSend* dependendo se todas as tarefas seriam para enviar para o mesmo utilizador ou não. Este processo apenas lê os valores dos resultados obtidos e executa o comando do *package mailutils:*

$ echo "message body" | mail -s "subject" test@example.com

- *"message body"* corresponde a um buffer que contém toda a informação adjacente á tarefa.
- *“subject”* é o valor do comando da tarefa.

Poderia também ser guardado em ficheiro com toda a informação da tarefa (*logfile* de execuçao de tarefas) e ser enviada toda a informação através do comando:

$ echo "%d" | sendmail test@example.com < taskLog.txt