C.M. DA COSTA*, S.S. TOSCANI**, T.B. CORSO***

SUMÁRIO

O artigo descreve a implementação da linguagem de comandos S-Shell do sistema operacional SONIX, que é compatível com a linguagem Shell do sistema operacional UNIX. São apresentadas todas as características da linguagem, sua gramática, os analisadores léxico e sintático, o gerador de código e o interpretador desse código.

ABSTRACT

This work describes the implementation of the command language S-Shell of the SONIX operatings system that is a subset os the UNIX shell language. The characteristics of the language, its grammar, the lexical and syntathical analysers, the code generator and the interpreter for this code are presented.

*B.SC. Análise de Sistemas (PUC-RS, 1978), aluno de mestrado em Ciência da Computação (UFRGS), professor assistente do Departamento de Ciências Estatísticas e da Computação (UFSC); áreas de interesse: sistemas operacionais e linguagens de programação; Curso de Pós-Graduação em Ciência da Computação da UFRGS; Av. Osvaldo Aranha, 99 - Porto Alegre - RS - CEP 90.000.

** Engenheiro Eletricista (UFRGS, 1967), Mestre em Informática (PUC/RJ, 1969); áreas de interesse: programação concorrente e sistemas operacionais; professor adjunto do Departamento de Informática/Curso de Pós-Graduação em Ciência da Computação da UFRGS; Av. Osvaldo Aranha, 99 - Porto Alegre - RS - CEP 90.000.

*** Engenheiro Civil (UFRGS, 1973), Mestre em Ciência da Computação (UFRGS, 1978); áreas de interesse: programação concorrente e sistemas operacionais; Professor Adjunto do Departamento de Informática/Curso de Pós-Graduação em Ciência da Computação da UFRGS; Av. Osvaldo Aranha, 99 - Porto Alegre - RS - CEP 90.000.

1. INTRODUÇÃO

S-Shell é a linguagem de comandos que serve de interface entre o usuário e o sistema Sonix. Esta linguagem é um subconjunto da linguagem Shell do Unix que mantém todas as principais características daquela.

Sua implementação foi feita por um programa escrito em Pascal Seqüencial. Este programa fica permanentemente sendo executado pelos processos Console do sistema operacional /COS 85a/.

A implementação da linguagem S-Shell foi realizada através da definição de uma máquina virtual (MV) que interpreta um conjunto de instruções apropriado para a implementação da linguagem S-Shell.

Os comandos dos usuários são transformados em instruções da máquina virtual, as quais são armazenadas em uma área de memória para posterior interpretação.

O interpretador de comandos foi estruturado em dois módulos, denominados Analisador e Executor.

No módulo Analisador são implementados os procedimentos necessários à análise léxica, a análise sintática e à geração de código para a máquina virtual. No módulo Executor foi implementado o procedimento que faz a interpretação do código gerado pelo módulo Analisador.

Os comandos executados pelo S-Shell podem ser oriundos em de terminal ou de um arquivo em disco. No segundo caso o usuário interativo, operando em um terminal, comanda a execução de uma outra ocorrência do programa S-Shell, redirecionando sua entrada para um arquivo construído previamente. Esta forma de uso do interpretador de comandos é denominada de "Procedimento S-Shell".

Quando o sistema Sonix é carregado ele cria um processo Console para cada terminal existente no sistema. Por sua vez, cada processo Console carrega para execução uma ocorrência do programa S-Shell, o qual passa a interagir com o usuário.

No momento em que o usuário digita uma linha de comando, esta linha é analisada e compilada para o código da máquina virtual. Se não é detectado nenhum erro, é disparado o processo de interpretação.

O S-Shell não é dotado de nenhum mecanismo de recuperação de erro. Quando for constatada a ocorrência de um, será enviada uma mensagem elucidativa para o usuário, todas as variáveis e estruturas de dados serão reinicializadas e o programa estará apto a aceitar um novo comando.

2. CONSTRUÇÕES DO S-Shell

O interpretador de comandos S-Shell possui as seguintes construções da linguagem Shell do Unix/RIT 78/:

Comandos Simples
Lista de comandos
Redirecionamento de entrada e saída
Linhas de montagem
Comando If
Comando While
Procedimentos Shell
Comandos em "background"

Algumas destas construções foram um pouco restringidas. Do comando If foi eliminada a cláusula Elseif; nos procedimentos Shell a passagem de parâmetros por assinalamento direto foi eliminada, sendo mantida a passagem de parâmetros posicionais. Com referência aos comandos em "background", somente é permitido que o caracter "&" que o indica apareça no final de um comando (simples ou lista de comandos). Assim, todos os comandos que precedem a tal caracter, no caso de existir mais de um, serão executados em background.

Em relação às variáveis Shell, permaneceram somente as que possuem significado implíci

to e que são mantidas e atualizadas pelo próprio Shell, e as que armazenam informações importantes para o usuário e que podem ser atualizadas pelo mesmo. As variáveis cujos conteúdos podem ser alterados pelo usuário são inicializadas pelo sistema no momento em que uma sessão é começada.

As variáveis anteriormente referidas são as seguintes:

Variáveis con Significado Implícito e Atualizáveis pelo Shell

$? - Contém o valor retornado pelo último comando executado. Este valor será zero se a execução tiver ocorrido normalmente e um no caso contrário.

$# - Armazena o número de parâmetros posicionais fornecidos quando a presente versão do Shell foi invocada.

$$ - Contém o número do processo associado a este Shell.

$! - Contém o número do último processo disparado no modo "background".

Variáveis Inicializadas no Momento em que Começa uma Sessão e Atualizáveis pelo Usuário

$Mail - Contém o nome do arquivo no qual o usuário deverá receber suas mensagens.

$Home - Contém o nome do diretório ao qual o usuário pertence.

$Path - Possui a lista de diretórios nos quais deverão ser procurados os arquivos contendo código executável. Se não forem especificados pelo usuário, "/Bin" e "/User/Bin" serão pesquisados. Na implementação do S-Shell é permitido ao usuário especificar até três diretórios.

As seguintes construções do Shell não foram implementadas:

- Comando Case
- Comando For
- Documentos locais
- Concatenação de saída
- Geração de nomes de arquivos
- Abolição do significado de metacaracteres

Alguns aspectos importantes, do ponto de vista de utilização, da linguagem S-Shell implementada são os seguintes:

a. Um comando simples pode ter um redirecionamento de entrada ("<"), um redirecionamento de saída (">"), ou ambos.

b. Nos comandos simples que fazem parte de uma linha de montagem ("|") não se pode ter no comando à esquerda do símbolo "|" uma alteração de saída e no comando à direita do "|" uma alteração de entrada.

c. Não pode haver redirecionamento múltiplo de entrada e saída.

d. O caracter "&", indicativo de "background", deve aparecer no final de um comando. Assim, por exemplo, em comandos if o caracter deve suceder ao Fi e em comandos While deve suceder ao Done.

3. GRAMÁTICA DA LINGUAGEM S-SHELL

A gramática será descrita com o uso da seguinte convenção:

- Os não terminais são escritos em letras maiúsculas.
- Os terminais são escritos em letras minúsculas.
- Uma produção será representada por um não terminal seguido de dois pontos e um conjunto de terminais e não terminais separados por brancos.
- Se várias produções começam pelo mesmo não terminal, serão postas em linhas separadas e o não terminal de origem da produção aparecerá somente uma vez.

```
ITEM : PALAVRA
       ENTRADA-SAÍDA

COMANDO-SIMPLES : ITEM
```

COMANDO-SIMPLES ITEM

```
COMANDO: COMANDO-SIMPLES
         (LISTA-DE-COMANDOS)
         While LISTA-DE-COMANDOS
         Do LISTA-DE-COMANDOS
         Done
         If LISTA-DE-COMANDOS
         Then LISTA-DE-COMANDOS
         Else LISTA-DE-COMANDOS
         Fi
         ATRIBUIÇÃO

LINHA-DE-MONTAGEM : COMANDO
                    LINHA-DE-MONTAGEM | COMANDO
EOU     : LINHA-DE-MONTAGEM
          EOU && LINHA-DE-MONTAGEM
          EOU || LINHA-DE-MONTAGEM

LISTA-DE-COMANDOS : EOU
                    LISTA-DE-COMANDOS;
                    LISTA-DE-COMANDOS&
                    LISTA-DE-COMANDOS; EOU

ENTRADA-SAÍDA : < PALAVRA
                >   PALAVRA

ATRIBUIÇÃO : NOME = PALAVRA

NOME : "seqüência de letras  começando por "$""

PALAVRA : "seqüência de letras e dígitos"

LETRAS : a, ..., z. $, /,-

DIGITOS : 0..9
```

As seguintes palavras são reservadas na linguagem:

If Then Else fi While Do Done

Os seguintes caracteres tem significado especial:

| Símbolo de linha de montagem
&& Símbolo de "E" lógico
|| símbolo de "OU" lógico
; Separador de comandos
& Comandos em "background"
() Agrupamento de comandos
< Redirecionamento de entrada
> Redirecionamento de saída

IMPLEMENTAÇÃO

A descrição da implementação do interpretador de comandos S-Shell será feita nas seções a seguir. Inicialmente será descrito o módulo Analisador e posteriormente o módulo Executor.

4.1 Módulo Analisador

O módulo Analisador é responsável pelo procedimentos de análise léxica, análise sintática e geração de código.

O analisador léxico opera sob demanda do analisador sintático, esse construído com o uso do método descendente recursivo.

Análise Léxica

A análise léxica é feita com o uso de seis procedimentos, os quais interagindo produ-

zem os elementos léxicos da linguagem, denominados "tokens". Os "tokens" são representados na seguinte estrutura de dados:

```
Type
    Classechar = (Letra, Dígito, Delimitador);
    Tipotoken  = Record
                      Classe  : Classechar;
                      Símbolo : Integer
                 End;
```

Os campos deste registro possuem o significado descrito a seguir.

Classe

Representa a classe a que o "token" pertence. Pode ser Letra, Dígito ou Delimitador.

Símbolo

Identifica o "token". Quando a classe a que o "token" pertence for letra, pode assumir os valores Palavra, Cif, Cthen, Celse, Cfi, Cwhile, Cdo ou Cdone. Quando a classe a que o "token" pertence for Dígito, assume o valor Número. Quando a classe for Delimitador pode assumir os valores Igual, E-comercial, Dois-ecom, Barra, Duas-bar, Ponto-virg, Abrepar, Fecha-par, Menor, Maior, Dois-pontos, Cr, Eof. Tratando-se de Palavra ou Número, o valor lido, que representa o nome de um comando ou o parâmetro para algum comando, encontra-se armazenado no vetor Nome.

A seguir serão apresentados os procedimento que implementam a análise léxica com a descrição da função que cada um desenvolve, individualmente.

Procedure Getchar;

Tem a função de ler um caracter de um dispositivo de entrada de dados e classificá-lo quanto a sua classe (Letra, Dígito ou Delimitador).

Procedure Classepal;

Verifica se a palavra contida no vetor Nome é uma palavra reservada da linguagem. Esta verificação é feita com o uso do comando Case.

Se a letra que ocupa a primeira posição no vetor Nome for inicial de alguma palavra reservada, então é verificado se o conteúdo do referido vetor é igual a esta específica palavra reservada. Se o resultado do teste for verdadeiro, significa que foi identificada uma palavra reservada. Neste caso a constante identificada por seu nome, acrescida do prefixo "C", será atribuído ao campo símbolo do registro que representa o "token".

Caso o resultado do teste seja negativo, o campo símbolo do referido registro receberá o valor Palavra, pois se está diante de um nome de comando ou nome de parâmetro.

Procedure Lepalavra;

Possui a função de ler uma seqüência de caracteres de um dispositivo de entrada de dados e armazená-los no vetor Nome.

Este procedimento é chamado sempre que, quando invocado pelo Léxico, o procedimento Getchar atribuir ao campo Classe do registro que representa o "token" o valor Letra, indicando com isto que uma letra pertencente ao alfabeto foi lida pelo mesmo.

Procedure Lenumero;

Sua função é idêntica à executada pelo procedimento Lepalavra, com a diferença de que é invocado para armazenar no vetor Nome uma seqüência de dígitos, os quais representam um número.

Procedure Saltabranco;

Este procedimento tem a função de saltar caracteres branco através de sucessivas chamadas ao procedimento Getchar.

Procedure Léxico;

É o procedimento básico de todo o processo de análise léxica. É chamado pelo analisador sintático sempre que esse necessita de um novo "token".

Através de sua ação, são disparados, apropriadamente, os demais procedimentos necessários para obter e tornar disponível um novo elemento léxico.

Análise Sintática

O analisador sintático é o gerenciador da interpretação das requisições de serviço feitas pelos usuários. Comanda a ação do analisador léxico, invocando-o sempre que necessita de um "token", e executa o atendimento da solicitação, através da chamada do Interpretador ao término da análise de um comando.

Na sua construção foi empregado o método descendente recursivo, o que permitiu explorar uma característica importante da linguagem Pascal, qual seja, a recursividade.

O código virtual gerado pelo analisador sintático é colocado em uma área de memória acessível ao interpretador (área de comunicação entre o analisador sintático e o interpretador). Todas as informações necessárias ao interpretador, geradas pelo analisador sintático, estão contidas no código, eliminando, desta forma, a necessidade de acessos a outras estruturas de dados.

Os procedimentos que implementam à análise sintática podem ser classificados em "Procedimentos de Análise e Geração de Código" e "Procedimentos Auxiliares".

Procedimentos de Análise e Geração de Código

Os procedimentos que se enquadram nesta categoria atuam de forma semelhante. Quando o usuário digita um comando ou quando o interpretador de comandos é acionado para executar comandos contidos em um arquivo, o programa S-Shell invoca o analisador léxico, o qual devolve um "token" da linguagem.

A partir deste "token" e, com o auxílio de um comando Case, a ação seguinte no processo de análise e geração de código é definida com a obtenção de um novo "token".

A geração de código é feita apropriadamente por alguns destes procedimentos através do procedimento Gera, o qual é invocado da seguinte maneira:

Gera (Mnem, Oper1 : Integer; Oper2: Identificador);

Os procedimentos anteriormente referidos são os seguintes:

Procedure Comando;
Procedure Listadecomandos;
Procedure Comandosimples;
Procedure Continuaçãodocomando;
Procedure Fimdocomando;
Procedure Instalaparametro;
Procedure Alteraentrada;
Procedure Alterasaída;
Procedure Atribuição;
Procedure Linhademontagem;
Procedure Andf;
Procedure Orf;

Procedimentos Auxiliares

Os procedimentos ditos Auxiliares são os que desenvolvem algum outro tipo de função necessária ao processo de análise e geração de código. Tais procedimentos são os seguintes:

Procedure Initvar;

Possui a função de inicializar todas as variáveis e tabelas, sempre que inicia o processo de interpretação de um novo comando.

Procedure Execcomandos;

É o procedimento que dispara o processo de interpretação. Fica permanentemente executando, somente deixando de fazê-lo quando o usuário encerra o S-Shell, através do comando Exit.

Procedure Atribui-par (Var a : Univ Identificador; i : Integer);

Atribui um parâmetro ao campo operando de uma instrução da MV. Sua existência é necessárias pois as variáveis Param a Instrução são de tipos diferentes.

Procedure Change-par;

A função desenvolvida por este procedimento é a de efetuar a substituição dos parâmetros posicionais pelos respectivos argumentos de chamada, fornecidos pelo usuário ao invocar a execução de um Procedimento S-Shell.

O usuário, quando está utilizando o sistema, o está fazendo através de uma ocorrência do programa S-Shell.

O S-Shell é um programa seqüencial como qualquer outro utilitário. Isto possibilita que o usuário comande a execução de uma nova ocorrência do mesmo, redirecionando sua entrada de dados para um arquivo criado anteriormente que contenha comandos do S-Shell. A isto denomina-se de Procedimento S-Shell.

Esta nova ocorrência do programa interpretador de comandos será executada por um novo processo.

Este processo, por sua vez, receberá do processo pai os parâmetros, se existirem, e os tornarão disponível e alteráveis ao programa seqüencial S-Shell através da declaração

Program P (var Param : Regparam)

suprida no prefixo do sistema Pascal.

A definição do tipo Regparam é a seguinte:

```
Type
    Identificador = Array (.1..20.) of Char;
    Regparam = Array (.0..9.) of Identificador;
```

Quando, durante a fase da análise e geração de código, o S-Shell identifica um "token" Eof, isto significa que o mesmo está atuando como um Procedimento S-Shell. Neste caso, antes de disparar a execução do módulo Executor, providência a substituição dos parâmetros posicionais por seus respectivos argumentos, invocando o procedimento Change-par, em questão.

Procedure Background;

Sua invocação se dá no momento em que é identificado um "token" Ecomercial.

A função desenvolvida por este procedimento é a de transferir o código virtual gerado até aquele momento para o monitor Controlproc, do sistema operacional Sonix, de onde o código será retirado para ser interpretado por um processo especialmente criado para este fim.

Com isto, o controle retorna imediatamente para o usuário que não precisará aguardar pelo término da execução das solicitações de serviço feitas anteriormente, podendo passar de imediato a enviar novos comandos.

Procedure Gera (Mnem, Operi : Integer; Oper2 : Identificador);

Sua função é gerar código para a MV. Recebe como parâmetros o mnemônico e o operando e produz uma nova instrução no vetor código.

4.2 Módulo Executor

As instruções geradas pelo módulo Analisador são executadas pelo módulo Executor.

A MV concebida para a implementação da linguagem de comandos, é constituída por uma memória que contém o código virtual gerado e por um Interpretador, responsável pela exe-

cução das instruções.

O Interpretador é invocado pelo procedimento "Comandos", pertencente ao analisador Sintático, sempre que um "token" Cr ou Eof, indicador de fim de comando do usuário, for encontrado.

Conjunto de Instruções

As instruções da MV são descritas a seguir:

JMP

O apontador de instruções virtuais (Pc) recebe o conteúdo do campo operando da instrução.

FALSEJMP

Pc recebe o conteúdo do campo operando da instrução se o último comando disparado não executou corretamente. Esta informação é obtida através da chamada do procedimento Resultexecp (I), pertencente ao sistema operacional.

COMMAND

Indica que um comando, cujo identificador está armazenado no campo operando da instrução, deve ser executado.

CPARAM

Contém, em seu campo Operando, o parâmetro do comando especificado em uma instrução Command. São geradas tantas destas instruções quantos forem os parâmetros fornecidos para o comando.

ALTERIN

A entrada de dados do comando a ser executado deve ser redirecionada para o arquivo especificado no campo operando desta instrução.

ALTEROUT

A saída de dados do comando deve ser redirecionada para o arquivo especificado no campo operando desta instrução.

CPIPE

Indica que deve ser estabelecido um canal de intercomunicação entre dois processos.

ASSIGN

Indica que deverá ser feita uma atribuição à uma variável S-Shell, do tipo atualizável pelo usuário, especificada no campo operando desta instrução.

VALUE

Deve suceder a uma instrução Assign. O conteúdo de seu campo operando é o valor que deve ser atribuido à variável especificada na última instrução Assign.

CWAIT

O processo que a executa fica bloqueado a espera que um processo filho termine. Isto é conseguido através da chamada do procedimento Waitp, pertencente ao sistema operacional.

No momento em que o processo filho termina sua execução, o processo que executou a presente instrução é sinalizado, o que ocasiona o seu prosseguimento. Esta instrução não possui operando.

CSTOP

Esta instrução ao ser executada faz com que o interpretador termine e o controle retorne ao módulo Analisador. A mesma não possui operando.

Exemplos de Geração de Código

```
Comando : Ls
Código Virtual Gerado : Command Ls
                        Cwait
```

```
                        Cstop
```

```
Comando : Cdr; Ls
```

```
Código Virtual Gerado : Command Cdr
                        Cwait
                        Command Ls
                        Cwait
                        Cstop
```

```
Comando : Ls > Meuarq
```

```
Código Virtual : Command Ls
                 Alterout Meuarq
                 Cwait
                 Cstop
```

```
Comando : Ed fonte < Arq
```

```
Código Virtual : Command Ed
                 Cparam Fonte
                 Alterin Arq
                 Cwait
                 Cstop
```

```
Comando : If Spascal Arqfonte Arqobjeto > Printer Then Arqobjeto
          Else
              Editor Arqfonte
          Fi
```

```
Código Virtual : 1 Command Spascal
                 2 Cparam Arqfonte
                 3 Cparam Arqobjeto
                 4 Alterout Printer
                 5 Cwait
                 6 Falsejmp 10
                 7 Command Arqobjeto
                 8 Cwait
                 9 Jmp 13
                10 Command editor
                11 Cparam Arqfonte
                12 Cwait
                13 Cstop
```

```
Comando : If Cdr Meudir
          Then Ls
          Fi
```

```
Código Virtual : 1 Command Cdr
                 2 Cparam Meudir
                 3 Cwait
                 4 Falsejmp 7
                 5 Command Ls
                 6 Cwait
                 7 Cstop
```

```
Comando : Ls | Who
```

```
Código Virtual : Command Ls
                 Cpipe
                 Command Who
                 Cwait
                 Cstop
```

```
Comando : Spascal Fonte Objeto && Objeto
```

```
Código virtual : 1 Command Spascal
                 2 Cparam Fonte
```

```
                3 Cparam Objeto
                4 Cwait
                5 Falsejmp 8
                6 Command Objeto
                7 Cwait
                8 Cstop
```

Comando : Spascal Fonte Objeto > Printer || Editor Fonte

```
Código Virtual : 1 Command Spascal
                 2 Cparam Fonte
                 3 Cparam Objeto
                 4 Alterout Printer
                 5 Cwait
                 6 Falsejmp 8
                 7 Jmp     11
                 8 Command Editor
                 9 Cparam Fonte
                10 Cwait
                11 Cstop
```

```
Comando : While Produztexto < Ent > Texto Do
          I     Imprimetexto < Texto > Printer
                Done
```

```
Código Virtual : 1 Command Produztexto
                 2 Alterin Ent
                 3 Alterout Texto
                 4 Cwait
                 5 Falsejmp 11
                 6 Command Imprimetexto
                 7 Alterin Texto
                 8 Alterout Printer
                 9 Cwait
                10 Jmp 1
                11 Cstop
```

Interpretador

A interpretação do código virtual gerado pelo módulo Analisador é feito por um procedimento denominado Interpretador.

Tal procedimento é formado por um comando de repetição (While) e por um comando de seleção (Case).

O comando While faz com que o procedimento permaneça em excução até que seja encontrada uma instrução Stop, que o encerra. No comando Case, cada rótulo identifica uma instrução e possui o procedimento necessário à sua execução.

Existem alguns comandos que, pela freqüência de uso, devem residir permanentemente na memória principal. Para tal, estes comandos foram codificados em um procedimento denominado Execbuiltin. Assim, no momento em que o Interpretador identifica que tem um comando para executar, ele deve primeiramente verificar se não se trata de um comando codidicado internamente. Esta verificação é feita através da invocação do procedimento Builtin. A técnica de pesquisa empregada é a mesma referida na descrição da análise léxica, para a identificação de palavras reservadas (utiliza-se o comando Case para a pesquisa em questão).

Acrescentar ou retirar comandos codificados internamente é algo trivial, pois significa incluir ou excluir entradas no comando Case e seus respectivos procedimentos.

Os comandos que foram codificados internamente são os seguintes:

Exit - Termina a execução do programa S-Shell.

Ps - Lista os processos existentes no sistema e os nomes dos programas que cada um está

tá presentemente executando.

Kill - Matá o processo especificado por seu número de criação (o qual é fornecido como argumento) e todos os seus descendentes.

5. CONCLUSÃO

O programa que implementa a linguagem S-Shell possui cerca de 1500 linhas de código fonte.

Sua depuração ainda não foi totalmente concluída porque os procedimentos que implementam redirecionamento de entrada, redirecionamento saida e "pipe" atuam diretamente nas tabelas que compõe o sistema de arquivos, e o mesmo encontra-se ainda em fase de desenvolvimento.

Do ponto de vista de utilização, a linguagem mostra-se fácil de ser usada e fácil de ser aprendida. Como a linguagem oferece ótimos recursos aos usuários, espera-se que a mesma venha a se constituir em uma ferramenta importante, no auxílio ao desenvolvimento de programas.

BIBLIOGRAFIA

/AVR 83/ AVRITZER, Alberto. µzunix um sistema operacional com suporte para multi-Processamento e tempo compartilhado. Belo Horizonte, UFMG, 1983. (dissertaçao de mestrado).

/BOU 78/ BOURNE, S.R. The Unix Shell. The Bell System Technical Journal, 57(6): 1971-90, July-Aug. 1978.

/BRA 84/ BRANDÃO, Maria de Fátima Ramos. Especificação de um sistema operacional multi-usuário em Pascal Concorrente. Porto Alegre, CPGCC/UFRGS, 1984. (dissertaçao de mestrado).

/CAR 81/ CARVALHO, M.A. de. Sistema Operacional Duo. Porto Alegre, CPGCC/UFRGS, 1981. (não publicado)

/COS 83/ COSTA, C.M.; BARBOSA, L.F.; FRIEDRICH, L.F.; SAYÃO, M. & CORSO, T.B. Descrição da implementação de um sistema operacional multiprogramado, escrito em Pascal Concorrente. IN: CONFERÊNCIA LATINO-AMERICANA DE INFORMÁTICA, 10., Viná Del Mar, Abril 1984. Anales. Viná del Mar, CLEI, 1984.

/COS 85/ COSTA, C.M. SONIX - Um sistema operacional multi-usuário escrito em Pascal Concorrente. Porto Alegre, CPGCC-UFRGS, 1985. (dissertaçao de mestrado).

/COS 85a/ COSTA, C.M.; TOSCANI, S.S. & CORSO, T.B. SONIX - Uma implementação do Unix em Pascal Concorrente. Porto Alegre, CPGCC/UFRGS, 1985 (Não publicado).

/DEI 84/ DEITEL, Harvey M. An introduction to operating system. Reading, Addison-Wesley, 1984.

/GRA 79/ GRAEF, N.; KRETSCHMAR, H.; LOER, K.P. & MORAWETZ, B. How design and implement small time-sharing systems using Concurrente Pascal. Software-Practice and Experience, 9(1): 17-24, Jan. 1979.

/HAN 77/ HANSEN, Per Brinch. The architecture of concurrent programs. Englewood Cliffs, Prentice-Hall, 1977.

/HOL 83/ HOLT, R.C. Concurrent Euclid, Unix System And Tunis. Reading, Addison-Wesley, 1984.

/ICH 83/ ICHIHARA, Ana Travassos. Projeto de um sistema de memória virtual para a máquina pascal concorrente emulada no computador ED-311. Porto Alegre, CPGCC/UFRGS, 1983. (Dissertaçao de mestrado).

/JOH 78/ JOHNSON, S.C. & RITCHIE, D.M. Portability of C programs and the Unix system. The Bell System Technical Journal, 57(6):2021-48, July-Aug. 1978.

/KRU 82/ KRUIJER, H.S.M. A Multi-user operating system for transaction processing, writen in Concurrent Pascal. Software-Practice and Experience, 12(5): 445-54, May 1982.

/MED 81/ MEDEIROS, Gil Carlos Rodrigues. Implementação do Sistema pascal Concorrente no computador Labo-8034. Porto Alegre, CPGCC/UFRGS, 1981. (dissertação de mestrado).

/RIT 74/ RITCHIE, D.M. & THOMPSON, K. The Unix time-sharing system. Communications of the ACM, 17/7):365-75, July 1974.

/RIT 78/ RITCHIE, D.M. & THOMPSON, K. The Unix-Sharing System. The Bell System Technical Journal, 57(6):1905-29, July-Aug. 1978.

/RIT 78a/ RITCHIE, D.M. Unix Time-Sharing System: A Retrospective. The Bell System Technical Journal, 57(6):1947-69, July-Aug. 1978.

/THO 78/ THOMPSON, K. Unix Implementation. The Bell System Technical Journal, 57(6):1931-46, July-Aug. 1978.

/WEB 82/ WEBER, T.S. & ROCHA, A.C. Descrição da arquitetura da máquina Pascal Concorrente. Porto Alegre, CPGCC/UFRGS, 1982. (relatorio interno).