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

SUMARIO

Este trabalho descreve a implementação do sistema operacional SONIX, compatível com o UNIX a nível de interface do usuário. O sistema foi escrito em Pascal Concorrente e um protótipo do mesmo encontra-se em funcionamento no minicomputador LABO-8034 do Curso de Pós-Graduação em Ciência da Computação da UFRGS.

ABSTRACT

This work describes the SONIX operating system which as UNIX compatible at the user interface level. The system was written in Concurrent Pascal and a prototype of it is already working on the LABO-8034 minicomputer of the Graduate Course in Computer Science at UFRGS.

* 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

Encontra-se em fase de conclusão, no Curso de Pós-Graduação em Ciência da Computação (CPGCC) da UFRGS, um projeto que tem por objetivo implementar a Máquina Pascal Concorrente (MPC), de forma microprogramada, no minicomputador ED-311.

No momento em que este projeto estiver concluído, todo o software disponível no sistema Pascal do LABO-8034 /MED 81/, incluindo-se os sistemas operacionais SOLO /MAN 77/ e DUO /CAR 81/, poderá ser transportado para o ED-311 sem problema algum.

Neste artigo é descrito um novo sistema operacional para a Máquina Pascal Concorrente que está sendo construido pelo grupo de pesquisa em Sistemas Operacionais do CPGCC/ /UFRGS. Este trabalho tem por objetivo oferecer melhores opções de uso para comunidade de usuários do CPGCC bem como desenvolver experiência na área de implementação de Sistemas Operacionais.

Na construção de um sistema operacional é necessário levar em conta dois aspectos muito importantes, quais sejam a especificação e a implementação do sistema especificado.

A decisão tomada foi o de implementar um sistema que já houvesse sido especificado anteriormente, de soluções inteligentes e que oferecesse aos usuários uma boa linguagem de comandos.

Um sistema com essas características e que, além disso, está se popularizando a ponto de muitos afirmarem tender a se tornar um padrão internacional é o UNIX, desenvolvido nos laboratórios da Bell, nos Estados Unidos /RIT 74/.

Assim, foi tomada a decisão de implementar um sistema em Pascal Concorrente, compatível com o UNIX a nível de interface do usuário, o qual foi chamado de SONIX.

2. DESCRIÇÃO DA IMPLEMENTAÇÃO DO NÚCLEO DO SISTEMA SONIX

A seguir será descrita a implementação do núcleo do Sonix. Seu detalhamento será feito em quatro partes. A primeira delas trata do controle de processos, a segunda da execução de programas sequenciais, a terceira da execução de procedimentos S-Shell e a quarta da inicialização do sistema.

2.1. Controle de Processos

No Sonix existem dois tipos de processos, processos Console e processos Envelope.

Os processos Console são os que interagem com o usuário; são criados quando da inicialização do sistema e executam o programa S-Shell que é o interpretador de comandos. Somente deixam de existir quando o sistema para de executar.

Processos Envelope, contrariamente, tem duração limitada. Um processo envelope é criado para executar um programa sequencial e "morre" após ter cumprido esta tarefa.

Como os processos Envelopes necessitam ser criados dinamicamente e na linguagem de implementação a criação de processos se dá de forma estática (o número de processos é definido em tempo de compilação), foi necessário criar um ambiente de simulação que permitisse tal coisa. A solução adotada foi empregar as primitivas de sincronização disponíveis na linguagem Pascal Concorrente, Delay e Continue, usadas nas estruturas monitor, para, juntamente com estruturas de dados e procedimentos, simular o ambiente necessário. Foi definido um monitor denominado Controlproc no qual foram implementados todos os procedimentos de sincronização e comunicação requeridos.

São mantidos no monitor Controlproc os registros descritores dos processos, bem como os procedimentos para criação e intercomunicação dos mesmos.

2.1.1 Criação de Processos

A criação de processos é feita da seguinte maneira: o processo pai executa o procedimento Fork do monitor Controlproc, o qual percorre os registros descritores de processos a procura de um que esteja no estado "D" (Desencarnado). Encontrado um nesta condição, seu estado é trocado para "V" (Vivo), sendo atualizados os demais campos do registro descritor. Após isto, o processo pai executa a operação Continue nesse processo, fazendo com que o processo filho passe a concorrer pelo uso do processador.

Registros Descritores de Processos

Cada processo é descrito no monitor Controlproc por um registro descritor, o qual é implementado pela estrutura de dados Desc que tem a seguinte constituição:

```
Type Estado = (V, D);
Type Desc   = Record
                    Número : Integer;
                    Atrib  : Integer;
                    Est    : Estado;
                    Idpai  : Integer
              End;
Var Descritor : Array (.0..Nproc.) of Desc;
```

Os campos deste registro possuem os seguintes significados:

Atrib : Contém a identificação do processo. E a mesma mantida no "kernel" da MPC, no registro descritor correspondente. Sua obtenção é feita pela instrução Attribute.

Est : Contém o estado do processo. Os valores possíveis são "V", significando que o processo esta "vivo", executando alguma ação, e "D" significando que o processo está "desencarnado", não estando executando nenhuma ação. Os processos Envelopes podem sofrer transição de um estado para outro. Os processos Console estão sempre no estado "V".

Idpai : Contém a identificação do processo pai.

2.1.2 Bloqueio e Sinalização de Processos

O procedimento padrão no Sonix é o processo pai criar um processo para executar um programa seqüencial e ficar a espera de que o processo filho termine a execução.

No momento em que isto ocorre, o processo pai é sinalizado pelo processo filho. Com isso o pai passa novamente a concorrer pelo uso do processador. Após ter sinalizado o processo pai, o processo filho "morre".

Primitiva Wait

O procedimento Waitm, do monitor Controlproc, faz com que o processo que o executa fique bloqueado a espera de sinalização. O bloqueio é feito quando o processo executa a instrução Delay no referido procedimento.

Primitiva Wakeup

O procedimento Wakeup, pertencente ao monitor Controlproc, faz com que o processo que o executa sinalize a seu processo pai, o qual passa a concorrer pelo processador. A sinalização é feita com a instrução Continue.

Primitiva Exit

O procedimento Exit, pertencente ao monitor Controlproc, faz com que o processo que o executa morra.

Isto é feito trocando o seu estado, no registro descritor, para "D". Após, isto o processo executa a operação Delay.

Primitiva Kill

O procedimento Kill, pertencente ao monitor Controlproc, permite que seja "morto" um determinado processo, especificado por seu número de criação. O estado do processo vítima é trocado para "D" e é executada a operação "Stop" no referido processo. Com isto, o programa seqüencial que o processo está executando será descontinuado na execução da próxima instrução virtual Falsejmp /MED 81/.

Todas as ações necessárias para "matar" o processo terão de ser efetuadas também sobre os seus descendentes, para que não sobrem processos "desgarrados" no sistema.

Nesta implementação da Kill o processo indicado sempre "morre", o que difere da implementação usual deste procedimento em outras versões Unix, nas quais o processo pode sobreviver /HOL 83/.

2.2 Execução de Programas Seqüenciais

Um programa seqüencial é executado transferindo seu código de disco magnético para a memória principal, colocando-o na área de dados do processo que irá gerenciar sua execução. Essa transferência é feita pela classe Loaderfile.

Existe também uma classe denominada Progstack que mantém as informações necessárias para permitir que programas seqüenciais possam chamar a outros programas sequenciais (chamadas aninhadas).

No Sonix existem três formas de execução para programas seqüenciais: através da criação de processos, sem envolver criação de processos e no modo "background". Estas três formas serão detalhadas a seguir.

2.2.1 Através da Criação de Processos

Esta é a forma básica usada pelo interpretador de comandos do sistema, o programa S-Shell. Quando o sistema é carregado, é criado um processo Console para cada terminal existente. O processo Console, por sua vez, carrega para sua execução o programa S-Shell, o qual fica recebendo comandos digitados pelo usuário. Sempre que o comando digitado implicar na execução de um programa seqüencial, será criado um processo para fazê-lo.

A criação do processo é realizada pelo procedimento Execpgm, o qual é invocado da seguinte forma:

Execpgm (Nameprog, Paramprog, Altera-IO, Recpipe);

Os parâmetros da chamada são definidos a seguir:

```
Type
    Identificador = Array (.1..20.) of Char;
    Regparam      = Array (.0..9.) of Identificador;
    Regaltera-IO  = Record
                          Ent, Sai : Boolean;
                          Nome-In, Nome-Out : Identificador
                    End;
    Pipetype = Record
                     Input, Output : Boolean;
                     Enderin, Enderout : Integer
               End;
Var
    Nameprog  : Identificador;
    Paramprog : Regparam;
    Alterá-IO : Regaltera-Io;
    Recpipe   : Pipetype;
```

Nameprog - Contém o nome do programa que deverá ser executado.

Paramprog - Informa os parâmetros que serão passados para o programa. Poderão ser no máximo dez.

Altera-IO - Contém informações sobre o redirecionamento de entrada e saída durante a execução do programa seqüencial. Os campos Ent e Sai indicam, respectivamente, alteração da entrada padrão e alteração da saída padrão. Nome-In e Nome-Out armazenam os nomes dos arquivos do redirecionamento.

Recpipe - Este parâmetro indica a criação de um canal de intercomunicação. Os campos Input e Output especificam para o processo se o mesmo vai depositar ou retirar dados do canal e Enderin e Enderout o endereço do canal através do qual os processos irão realizar a troca de mensagens.

O procedimento Excepgm cria um processo através da chamada do procedimento Fork do monitor Controlproc. O procedimento Fork devolve o número da entrada no array de registros descritores que descreve o processo criado. A seguir, Execpgm chama o procedimento Execpgm-M, do monitor Controlproc e passa todos os parâmetros recebidos quando da sua chamada e mais o número da entrada que descreve o processo criado.

O procedimento Execpgm-M armazena na estrutura de dados denominada. "Execute" todas as informações recebidas e em "Tabwork" a indicação de que o processo criado tem um programa seqüencial para executar. A estrutura Execute possui a seguinte definição:

```
Type Regexec = Record
                    Seqprog   : Identificador;
                    Paramprog : Regparam;
                    Recpipe   : Pipetype;
                    Altera-IO : Regaltera-IO
               end;
Var Execute : Array (.1..Nproc.) of Regexec;
```

O processo criado, após identificar que tem um programa seqüencial para executar, chamará o procedimento Fetchpgm, pertencente ao monitor controlproc, o qual simplesmente devolverá o conteúdo da estrutura de dados Execute. Após isso, o processo efetuará a chamada do procedimento Runpgm que, com o auxílio das classes Loaderfile e Progstack fará com que o programa seqüencial seja executado.

2.2.2 Sem Criação de Processos

Um programa qualquer em execução pode disparar um outro programa através da chamada do procedimento Execv.

Este procedimento tem a função de efetuar a chamada Runpgm passando as informações recebidas. A partir dai tudo se passa como descrito na seção anterior.

Como trata-se de uma chamada aninhada de programa, após o término do programa chamado, a execução deverá retornar ao programa que efetuou a chamada; para tanto o mesmo será recarregado e será restaurado seu contexto.

2.2.3 Modo "Background"

Basicamente, a função do S-Shell é a de ler, analisar e executar um comando. A leitura e a análise são realizadas pelo módulo Analisador enquanto a execução é realizada pelo módulo Executor. Quando a execução do comando implica na execução de um programa sequencial, o processo Console, que está executando o S-Shell, cria um processo filho para executar tal tarefa e espera pelo término do mesmo. Somente após isso volta a aceitar novos comandos do terminal.

Esta situação será alterada se o usuário disparar a execução de um comando em "background": o processo Console passará imediatamente a aceitar novos comandos do usuário, sem esperar pelo término da execução do comando anterior.

A ação de executar um comando em "background" pode ser dividida em duas fases. Na fase 1, quando o módulo Executor do programa S-Shell reconhece que o código virtual gerado pelo módulo Analisador deve ser executado em "background", chama ao procedimento Execbcg pertencente ao processo Console, passando como parâmetro o referido código. Esse procedimento cria um processo e faz a chamada Execbcg-M, pertencente ao monitor Controlproc, que tem a função de depositar na estrutura de dados "Background" o código gerado pelo programa S-Shell que deve ser interpretado pelo processo criado. Essa estrutura de dados possui a seguinte definição:

```
Const Lengthbcg = 20;
Type
    Tipooperando = (Inteiro, Caracter);
    Instrução    = Record
                            Flag: tipooperando;
                            Mnemonico : Integer;
                            Operando  : Identificador
                       End;
    Codetype = Array (.1..Lengthbcg.) of Instrução;
    Typebackground = Record
                          Code : Codetype;
                          Free : Boolean
                     End;
Var Background : Typebackground;
```

O campo Code armazena o código virtual a ser executado e o campo Free indica se a estrutura de dados está disponível ou não. Quando o código é armazenado Free recebe o va

lor False.

Imediatamente após o término do procedimento Execbcg o processo Console retorna ao contexto do programa S-Shell, estando apto, dessa forma, a aceitar novos comandos do terminal.

Na Fase2 o processo criado, ao identificador que deve executar um código em modo "background", chama o procedimento Fetchbcg, pertencente ao monitor Controlproc, que lhe devolve o código armazenado na estrutura de dados "Background" e torna essa estrutura disponível, (fazendo o campo Free receber o valor True).

Feito isso, dispara a execução do interpretador da linguagem de comandos (módulo Executor), o qual foi codificado como um procedimento dos processos. A partir desse momento é iniciada a interpretação do código virtual.

Sempre que for necessário, será criado um processo filho para a execução de um programa sequencial. Quando isso ocorrer, as coisas se passarão como descritas na primeira forma de execução de programas sequenciais que foi apresentada (através da criação de processos).

2.3 Procedimentos S-Shell

O processador da linguagem de comandos, denominado S-Shell foi programado como um programa seqüencial. Quando o sistema é inicializado, cada processo Console o carrega para a memória principal e começa a executá-lo.

O usuário, interagindo com o sistema através do S-Shell, poderá comandar a execução do programa S-Shell no modo "background" redirecionando sua entrada para um arquivo anteriormente criado e que contenha instruções S-Shell.

No referido arquivo poderão aparecer parâmetros posicionais referidos como $1,$2,..$n que serão substituídos pelos argumentos fornecidos pelo usuário ao disparar a execução de um Procedimento S-Shell, substituição esta que será feita antes de ser invocado o módulo Executor do processador da linguagem.

Como os procedimentos S-Shell são executados no modo "Background", o usuário continuará normalmente no terminal interagindo com a ocorrência (instanciada) do S-Shell carregada quando da inicialização do sistema.

2.4 Inicialização do Sistema

Quando o programa concorrente que implementa o sistema Sonix é carregado, sua execução começa pelo processo Ancestor (codificado como processo inicial na linguagem Pascal Concorrente), o qual efetua a operação "Init" no monitor Controlproc e, a seguir, nos processos do sistema (processos Sist), passando o referido monitor como parâmetro. A partir disso, todos os processos passam a concorrer pelo processador.

Cada processo Sist inicializa as classes Stackprog e Loaderprog e chama o procedimento Setatrib, pertencente ao monitor Controlproc. O procedimento Setatrib coloca no registro descritor do processo a sua identificação, a qual é obtida do "kernel" através da instrução Atributte. Ao término deste procedimento o processo "se mata" chamando o procedimento Exit do monitor em questão.

A partir disto, do ponto de vista do Sonix, somente existem o processo Ancestor e o monitor Controlproc. Na realidade existem "Nproc" processos do tipo "Sist" em delay no monitor Controlproc, esperando para serem criados" quando então desempenharão os papéis de processos Console ou processos Envelope.

No passo seguinte o processo Ancestor acessa o monitor Controlproc e configura o sistema, criando um processo console para cada terminal. O número de terminais é indicado pela constante Noterm, a qual pode ser alterada para refletir uma mudança no número de terminais da configuração.

Após ter realizado este trabalho, o processo Ancestor deixa de existir pois não tem mais nenhuma função a realizar.

Os processos Consoles criados carregam e começam a executar o programa S-Shell, passando dessa, forma, a tornar o sistema disponível aos usuários. Os processos envelopes serão "criados" pelos processos Consoles, durante a execução do S-Shell, para executarem comandos do usuário.

2.5 Estrutura dos Processos Sist

Um processo Sist pode tanto desempenhar o papel de um processo console como de um processo Envelope. O corpo principal dos processos Sist é o seguinte:

```
Begin
     Init Stackprog, Loaderprog;
     Estadoproc.Setatrib;
     Estadoproc.Exit;
     Cycle
          If Estadoproc.Myfunction = Console Then
          Begin
               Idok := False;
               While Not Idok do
                    Execident (Idok);
               With Execute do
               Begin
                    Seqprog := 'S-Shell'
                    Altera-io.Ent := False;
                    Altera-io.Sai := False;
```

```
                Recpipe.Input  := False;
                Recpipe.Output := False;
            End;
            Runpgm (Execute, Line, Result);
        End
        Else
        Begin
            Mywork := Indefined;
            While Mywork = Indefined Do
            Begin Estadoproc.Mywork;
                Wait;
            End;
            Case Mywork of
                Back : Begin
                         Estadoproc.Fetchbcg (Cod-back)
                         Interpretador (cod-back)
                       End;
            Procshell,
                Prog : Begin
                         Estadoproc.Fetchbcg (Executei)
                         Runpgm (Execute, Line, Result)
                       End
            End;
            Start;
            Estadoproc.Wakeup;
            Estadoproc.Exit;
        End;
    End;
End;
```

Quando o processo Ancestor configura o sistema, ele cria um processo Console (isto é, um processo Sist com a função de Console) para cada terminal existente.

Um processo Console ao receber o processador, identifica sua função e tenta reconhecer um usuário válido através do procedimento Execident.

Ao identificar um usuário no terminal, este processo prepara os parâmetros de forma adequada e dispara a execução do programa S-Shell através do procedimento Runpgm.

No momento em que o usuário encerrar sua sessão, terminando a execução do S-Shell, o processo console tentará a identificação de um novo usuário, repetindo, dessa forma, o procedimento descrito.

Os processos Console existirão enquanto o sistema Sonix estiver operando.

O sistema Sonix suporta "criação dinâmica" de processos para execução de programas seqüenciais, o que é requerido para implementar a linguagem de comandos disponível aos usuários.

Conforme já referido anteriormente, criar um processo significa chamar o procedimento Fork, do monitor Controlproc, o qual irá procurar nos registros descritores de processos pela existência de um no estado "D". Encontrando, colocará no registro descritor as informações adequadas e efetuará a operação Continue no processo correspondente, para que ele passe a executar a função de Envelope.

Como um processo Envelope é criado e começa a disputar o processador antes das estruturas de dados que informam o trabalho que o mesmo terá de realizar estarem atualizadas, ele fica em um laço chamando o procedimento Estadoproc.Myfunction, do monitor Controlproc, que lhe fornece esta informação. A instrução Wait, codificada neste laço faz com que o processo que a executa deixe de concorrer pelo processador por um pequeno espaço de tempo (um segundo); assim, tenta-se agilizar a atualização das estruturas de dados referidas.

O processo Envelope é capaz de realizar três tipos de trabalho. Se Mywork for igual a Back, o processo envelope executará um código em "background". Para isto, chamará o procedimento Estadoproc.Fetchbcg, pertencente ao monitor Controlproc, que lhe devolverá o código a ser executado. A execução desse código será feita através da chamada do módulo Executor. Se Mywork for igual a Procshell ou igual a Prog, o processo Envelope executará um programa seqüencial. Neste caso, chamará o procedimento Estadoproc.Fetchpgm, pertencente ao monitor Controlproc, que lhe devolverá o nome do programa a ser executado bem como seus parâmetros. A execução se processará através da chamada Runpgm, passando as informações recebidas. Quando se tratar de Procshell, o programa sequencial em questão será o S-Shell. Neste caso, antes de ser disparado o módulo Executor, se houverem parâmetros, os mesmos terão de ser substituidos.

Após o término de sua tarefa (Back, Prog ou ProcShell), o processo Envelope executará a instrução Start, possibilitando, dessa forma, a execução de novos programas seqüenciais (caso o último tenha sido descontinuado por uma instrução Stop). Chamará então o procedimento Estadoproc.Wakeup para acordar seu pai, que poderá estar aguardando pelo término de sua execução, e, finalmente, "morrerá", executando o procedimento Estadoproc.Exit.

3. CONCLUSÃO

O sistema SONIX deverá ser compatível com o UNIX, também, a nível do sistema de arquivos. Entretanto, como a implementação do sistema de arquivos projetada estava um tanto atrasada, decidiu-se incorporar ao SONIX o sistema de arquivos do sistema SOLO. Is-

to permitiu colocar em funcionamento em protótipo da SONIX no minicomputador LABO-8034. Este propótipo está operacional desde março do corrente ano.

O programa Pascal Concorrente que implementa o sistema SONIX possui cerca de 1500 linhas de código fonte. Foram efetuados testes metódicos e exaustivos, o que permite ter-se uma grande confiança na correção do seu funcionamento.

Paralelamente com os testes e com a implementação do sistema de arquivos estão sendo preparados os utilitários do sistema. Desta forma espera-se que em um pequeno espaço de tempo se possa colocar à disposição dos usuários em sistema completo, com grandes facilidades para o desenvolvimento de suas aplicações.

A experiência adquirida no desenvolvimento deste trabalho permite que se enfatize a conveniência de usar-se linguagens de alto nível no desenvolvimento de sistemas operacionais. A implementação destes sistemas complexos fica enormemente facilitada com a utilização de uma linguagem adequada.

BIBLIOGRAFIA

/AVR 83/ AVRITZER, Alberto. µzunix um sistema operacional com suporte para multiprocessamento e tempo compartilhado. Belo Horizonte, UFMG, 1983. (dissertação 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. (dissertação 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 LATINOAMERICANA de INFORMÁTICA, 10., Vina Del Mar, Abril 1984. Anales. Vina del Mar, CLEI, 1984.

/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 Concurrent 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. (Dissertação 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 Time-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. (relatório interno).