UDK 681.3.014

Vido Vouk
Jure Ferbežar
Andrej Brodnik
Institut »Jožef Stefan«

This article presents a multi tasking environment for the MS-DOS operating system on the IBM-PC computer. Real time scheduler was developed using real time clock interrupts to perform process scheduling. Interprocess communication is implemented using semaphores and message exchange. All the support routines for the multi-tasking real-time environment are packed in a single module which offers all the routines needed to handle process manipulation, process synchronization and interprocess data exchange. All the software is developed using Logitech Modula-2 environment.

Članek opisuje osnovno okolje za pisanje večopravilnih programov na računalniku IBM-PC pod operacijskim sistemom MS-DOS. V ta namen smo razvili razporejevalnik procesorskega časa, ki uporablja urine prekinitve. Za medprocesno sinhronizacijo smo uporabili semaforje in izmenjavo sporočil. Uporabniku je na voljo modul podprogramov, ki nudi vse potrebne podprograme za delo v večprocesorskemu okolju. V modulu so podprogrami za ustvarjanje, poganjanje, ustavljanje in izločanje ter podprogrami za nadzor procesov, podprogrami za komunikacijo med procesi (send, receive) in podprogrami za sinhronizacijo (wait, signal) in ustrezni dodatni podprogrami za ustvarjanje, nadzor in izločanje semaforjev. Vsa programska oprema je napisana v programskem jeziku modula-2.

1. UVOD

Računalniki združljivi z IBM-PC so pri nas vedno pogostejši in tudi niso pretirano dragi. Žal pa ti računalniki ne nudijo podpore v večopravilnem okolju. Ker se zaradi nizke cene vedno več potrošnikov odloča za ta tip računalnika, so razvijalci prisiljeni poiskati, oziroma izdelati čimbolj univerzalna orodja za izdelavo zahtevnejših aplikacij. V tem članku predstavljeni razporejevalnik procesorskega časa predstavlja eno od takih orodij. Žal zaradi omejenosti operacijskega sistema MS-DOS iz fička ni mogoče narediti Ferrarija.

1.1 PROCESI

Proces lahko definiramo kot asinhrono aktivnost, naprimer izvajanje programa na centralni procesorski enoti CPE /HOL78/. S preprostim razmislekom lahko pridemo do zaključka, da je proces lahko v poljubnem trenutku opazovanja le v enem od dveh možnih stanj

IZVRŠLJIV ------------ NEIZVRŠLJIV

Opisani stanji lahko zaradi lažjega razmišljanja razdelimo naprej na naslednja podstanja:

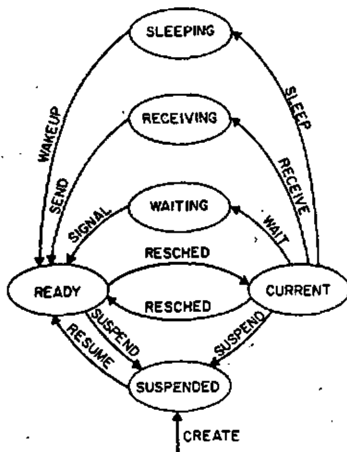
IZVRŠLJIV	- TRENUTNI
	- PRIPRAVLJEN
NEIZVRŠLJIV	- PREKINJEN
	- ČAKAJOČ
	- SPEČ
	- SPREJEMAJOČ

Privzemimo, da imamo več procesov. Vsi procesi iz skupine IZVRŠLJIV imajo pravico do izvajanja na poljubnem CPE. V poljubnem trenutku pa se dejansko izvaja eden (na eno procesorskem sistemu) ali N procesov (na N procesorskem sistemu). Procese, ki se izvajajo, imenujemo TRENUTNI. Vsi ostali procesi, ki imajo pravico do izvajanja, pa se ne izvajajo zaradi pomanjkanja prostih procesorjev, so v stanju PRIPRAVLJEN.

Procesi iz skupine NEIZVRŠLJIV so začasno ustavljeni in nimajo pravice do procesorja, čeprav bi kak procesor bil prost ali celo brez dela. Ti procesi so ustavljeni in čakajo na nek zunanji dogodek, ki jih zbudi in jih prestavi v skupino IZVRŠLJIVIH procesov. Dogodki, ki jih zbudijo in prestavijo v drugo skupino, so različni glede na stanje v katerem so zaustavljeni.
Slika 1.1 prikazuje možna stanja procesov in dogodke, ki vplivajo na prehode med stanji.
K sliki 1.1 moramo dodati tudi kratek komentar. Iz slike je razvidno, da klic podprograma CREATE ustvari nov proces in ga postavi v stanje PREKINJEN. Vprašanje, ki se pojavi je, kaj se zgodi s procesom, ki ga želimo odstraniti iz našega okolja. Klic KILL odstrani proces ne glede na to kje se je nahajal pred klicem. Proces nepreklicno izgine iz okolja. Pred odstranitvijo sprosti vse zasedene zmogljivosti v sistemu.

1.2 SINHRONIZACIJA

Za sinhronizacijo procesov v večprocesornih okoljih uporabljamo različne tehnike. Dokažemo lahko, da so vse tehnike funkcionalno enakovredne /FIL84/, vendar so v



Slika 1.1
Stanja procesov in prehodi med stanji

različnih izvedbah različno primerne. Razporejevalnik opisan v tem članku za sinhronizacijo uporablja semaforje /DIJ75/ in izmenjavo sporočil /JON87/.

1.2.1 SEMAFORJI

Kot sta pokazala /DIJ75/ in /HAN73/ lahko vse probleme sinhronizacije rešimo s P in V semaforjema. V naši izvedbi razporejevalnika smo uporabili splošne (števne) semaforje. Torej je binarni semafor v našem razporejevalniku le posebna oblika splošnega semaforja.

1.2.2 IZMENJAVA SPOROČIL

Izmenjava sporočil (message exchange) je poseben mehanizem, ki omogoča procesu da prenese pripravljeno sporočilo drugemu procesu. Izmenjava sporočil je obenem poseben mehanizem sinhronizacije. Glavna razlika med semaforji in izmenjavo sporočil je v tem, da mora biti za vsak klic WAIT(sem) ustrezen klic SIGNAL(sem), v primeru izmenjave sporočil pa to ni nujno. Izmenjava sporočil je za sinhronizacijo preprostejša posebno kadar proces ne ve vnaprej koliko sporočil bo dobil in kateri procesi jih bodo poslali. Ta način sinhronizacije je uporabljen v CSP /HOA85/, moduli-2 /WIR85/ in v programskem jeziku OCCAM /JON87/. Dokažmo lahko, da je možno izvesti izmenjavo sporočil samo s semaforji in obratno /FIL84/.

1.3 RAZPOREJANJE PROCESOV

Naša nadaljna razlaga bo zadevala enoprocesorske sisteme. Posplošitev na večprocesorske sisteme je očitna.
Iluzijo vzporednega izvajanja več procesov na eno ali večprocesorskemu sistemu lahko dosežemo s preklapljanjem procesorja (procesorjev) med več procesi.
Če hočemo zagotoviti, da bodo procesi enakopravno izkoriščali dane procesorske zmogljivosti v sistemu, moramo uvesti razsodnika, ki odloča, kdaj bo kateri od procesov začel z izvajanjem, kdaj bo izvajal svojo kodo in kdaj bo prenehal z delom, da bi prepustil procesor naslednjemu uporabniku. Ta razsodnik je razporejevalnik procesorskega časa. Razporejevalnik odloča kateri proces lahko zasede procesor in ga potem tudi lahko prekine, da dodeli procesor drugemu procesu. Vsakemu procesu določimo nek določen čas, ko lahko teče, ne da bi ga prekinili. Po izteku tega časa damo v izvajanje na tem procesorju nek drug proces.

2. IZVEDBA

2.1 PROGRAMSKO OKOLJE

Razporejevalnik časa smo razvili za IBM-PC združljiv računalnik. Za modulo-2 smo se odločili, ker je višji programski jezik /WIR85/, ki ima vse potrebne konstrukte za delo na strojnem nivoju in obenem osnovno podporo za delo z več procesi. Tako je razvoj in vzdrževanje programske opreme relativno preprosto. Vsa programska oprema razporejevalnika je napisana v Logitech Moduli-2/86 pod operacijskim sistemom MS-DOS 3.20.

2.2 UVOD

Razporejevalnik je zasnovan na primeru razporejevalnika za RT-11 /BRO87/. Za razliko podobnih razporejevalnikov, ki delujejo sinhrono /COL87/, naš razporejevalnik omogoča delo v realnem času. Informacijo o pretečenem času dobi od prekinitev, ki jih generira ura. Urine prekinitve so relativno pogoste. Če bi razporejevalnik ob vsaki urini prekinitvi pregledal vse strukture, bi bil odziv sicer izjemno dober, vendar bi bila učinkovitost takega sistema zelo majhna, saj bi se tak sistem večino časa ukvarjal sam s seboj. Tako ob vsaki urini prekinitvi razporejevalnik pregleda le nekatere strukture, medtem ko splošni pregled stanja procesov naredi le na določeno število urinih prekinitev. Uglaševanje izvedemo eksperimentalno.
Razporejevalnik odloča, kateri od procesov bo dobil pravico do izvajanja svoje kode. Razporejevalnik ga tudi prekine med izvajanjem in to tako, da se proces tega ne zaveda. Razporejevalnik shrani vse potrebne podatke o procesu v lokalni pomnilnik, tako da lahko v poljubnem trenutku ponovno zažene prekinjeni proces. Proces nadaljuje z izvajanjem v točki v kateri je bil prekinjen.

Med delom smo naleteli na več težav. Največjo težavo je predstavljala sama zasnova operacijskega sistema. MS-DOS 3.20 je eno uporabniški eno opravilni operacijski sistem, tako da nobeden od sistemskih klicev ni prekinljiv. To težavo smo poskušali zaobiti z uporabo nedokumetirane lastnosti operacijskega sistema /LOG85/, ki s posebno zastavico označi kdaj je v kritičnem odseku. Vendar se ta rešitev med izdatnim testiranjem ni izkazala kot dovolj zanesljiva, ker tudi nekateri podprogrami v moduli-2 niso prekinljivi. Ta problem smo rešili tako, da smo celotni klic DOSa označili kot kritični odsek.

Celoten sistem je sestavljen iz treh funkcionalno ločenih modulov: jedro, semaforji in izmenjava sporočil. Jedro omogoča delo nad procesi (ustvarjanje, prekinitev procesa), modul semaforjev omogoča delo s semaforji (ustvarjanje, SIGNAL, WAIT) modul za izmenjavo sporočil pa omogoča klice SEND in RECEIVE. Ker je naš sistem namenjen tudi uporabi v povsem realnih aplikacijah, so vsi trije moduli skriti v oklepajočem modulu, ki zunanjemu uporabniku onemogoča dostop do kritičnih ukaznih in podatkovnih struktur. Iz modula navzven so iznešene le funkcije, ki so potrebne za učinkovito uporabo sistema. Uporabnik nima nobene možnosti, da bi z

nepravilno ali nepazljivo uporabo porušil konsistentnost sistema. V dodatku A je podan definicijski modul našega razporejevalnika, iz katerega je razvidno, katere podatkovne in ukazne strukture ter klici so uporabniku na voljo.

2.3 PODATKOVNE STRUKTURE

Za učinkovitejše delo razporejevalnika smo morali dobršen del prekinitvenih podprogramov zamenjati s svojimi novo napisanimi, ki uporabljajo konstrukte iz razporejevalnika. Podatkovne strukture izven modula niso vidne /BRO87/. Osnovne enote, s katerimi uporabnik lahko upravlja, so procesi, semaforji in sporočila. Z njimi operira preko klicev podprogramov in tako vpliva na tek procesov. Vsi podprogrami so monitorji /HAN75/.

2.3.1 PROCESI

Osnovna enota v sistemu je proces. Proces je v moduli-2 izvajanje podprograma brez parametrov /WIR85/ in /LOG86/. V našem sistemu smo definicijo razširili z naslednjimi parametri :

- ime
- interno ime
- življenjski prostor
- prioriteta

Ime je sestavljeno iz niza alfanumeričnih znakov. Namenjeno je le uporabniku. Ob zahtevi za ustvaritev procesa jedro preveri pravilnost parametrov in ustvari proces. Priredi mu interno ime, ki ga vrne uporabniku. Vsi nadaljni klici za delo z ustvarjenim procesom uporabljajo le interno ime. življenski prostor opredeljuje velikost procesu prirejenega delovnega pomnilnika /LOG86/. Prioriteta je pomemben parameter, ki opisuje nujnost izvajanja danega procesa. Večja vrednost predstavlja večjo prioriteto. V osnovni izvedbi našega sistema velja, da se vedno izvaja tisti proces, ki ima najvišjo prioriteto. Kadar je takih procesov več, si ti procesi enakopravno delijo procesor med seboj (round robin). Naslednje izvedbe sistema omogočajo drugačne načine razporejanja /VMS82/.
Vsak proces opredeljuje spremenljivka stanja (prim. slika 1.1). Kot smo omenili, se lahko proces v danem trenutku nahaja le v natanko enem stanju. Kratek opis stanj:

TRENUTNI (CURRENT): Ker je IBM-PC enoprocesorski sistem je lahko v poljubnem trenutku le en TRENUTNI proces. Ta proces izvaja svojo kodo. če sam ne kliče nobenega od podprogramov, ki bi mu lahko spremenil status (WAIT, RECEIVE, SLEEP, SUSPEND), ga prekine razporejevalnik ko poteče njegov interval časa, ali pa če se prebudi kak proces z višjo prioriteto. Kadar je ta proces edini s tako prioriteto, bo ponovno dobival pravico do izvajanja po en interval, dokler ne bo končal ali pa zamenjal statusa.

PRIPRAVLJEN (READY): V tem stanju so vsi procesi, ki imajo vse pogoje za izvajanje in čakajo le na prost procesor.

PREKINJEN (SUSPENDED): Po klicu za ustvaritev razporejevalnik ustvari proces in ga postavi med NEIZVRŠLJIVE procese. Proces ostane v tem stanju, dokler ga ne obudi eden od TRENUTNIH procesov. Razporejevalnik lahko v to stanje postavi tudi poljuben PRIPRAVLJEN proces, če to od njega zahteva kateri izmed procesov.

ČAKAJOČ (WAITING): Razporejevalnik postavi poljuben TRENUTNI proces po klicu WAIT v to stanje, če je vrednost semaforja ob klicu nič. Proces ostane v tem stanju dokler ga s klicem SIGNAL ne obudi eden izmed TRENUTNIH procesov.

SPEČ (SLEEPING): Proces postavi v to stanje razporejevalnik po klicu SLEEP. Iz tega stanja ga spet obudi razporejevalnik po izteku zahtevanega časovnega intervala.

SPREJEMAJOČ (RECEIVING): Po klicu RECEIVE proces preide v to stanje, če zanj še ni prispelo nobeno sporočilo in ostane v tem stanju dokler ne pride sporočilo od enega izmed TRENUTNIH procesov.

2.3.2 SEMAFORJI

Semaforji v našem sistemu predstavljajo osnovni princip sinhronizacije. Uporaba in delovanje so obširno opisani v literaturi (npr. /HAN73/). V naši izvedbi razporejevalnika uporabljamo splošne semaforje. Tip semaforja določimo s klicem podprograma za ustvaritev semaforja.
Vsak klic SIGNAL (P) poveča vrednost semaforja za 1 če je vrsta čakajočih na ta semafor prazna, sicer pa za vsak SIGNAL obudi enega od čakajočih.
Po vsakem klicu WAIT (V) razporejevalnik pogleda ali je števec semaforja večji od nič. če ni, uvrsti klicajoči proces v vrsto čakajočih na dani semafor, sicer ga pusti naprej v izvajanje in popravi števec semaforjev. Princip čakanja je po sistemu FIFO /WIR76/.

2.3.3 IZMENJAVA SPOROČIL

Razporejevalnik poleg semaforjev omogoča tudi izmenjavo sporočil. čeprav lahko naredi izmenjavo sporočil uporabnik sam s semaforji, smo se odločili, da jo izvedemo še posebej, tako da zmanjšamo dodatno delo na najmanjšo možno mero.
Pošiljanje sporočil je izvedeno tako, da proces, ki pošilja sporočilo, ne preneha z izvajanjem tudi kadar ne more oddati sporočila. če sporočila ni mogoče oddati, dobi o tem ustrezen odgovor, vendar se njegovo stanje (TEKOČI) zaradi tega ne spremeni.
če je proces že dobil kako sporočilo, pa ga še vedno ni prevzel, so vsa nadaljnja pošiljanja temu procesu neveljavna vse dokler ga ne prevzame. Na ta način preprečimo, da bi se sporočila kopičila in porabila ves razpoložljivi prostor. Osnovna izvedba razporejevalnika omogoča pošiljanje naslova. Ker mora biti prostor za sporočilo izven prostora pošiljatelja (podiranje sklada), je prostor za sporočilo v jedru razporejevalnika.

2.4 SPEČI PROCESI

Včasih je potrebno, da proces sinhrono odstopi procesor za točno določen časovni interval. V ta namen je na razpolago poseben klic, ki postavi proces v vrsto spečih procesov. Za take procese je v razporejevalniku izdelana posebna struktura, ki procese razvrsti po naraščajočem času čakanja. časi čakanja so podani relativno glede na predhodnika. Tako razporejevalnik ob vsaki urini prekinitvi pogleda le prvi proces v vrsti in ga prebudi, če je njegov čas spanja že potekel.

2.5 STRATEGIJA RAZPOREJANJA

V osnovni izvedbi razporejevalnika je strategija razporejanja zelo preprosta. Procesi se med sabo lahko razlikujejo po pririteti. Izvaja se le en proces in to proces z najvišjo prioriteto. Kadar je teh procesov več, se med seboj izmenjujejo in si enakovredno delijo procesorski čas. Procesi z nižjo prioriteto čakajo na izvajanje vse dokler vsi procesi z višjo prioriteto ne preidejo v eno izmed stanj skupine NEIZVRŠLJIV ali ne zapustijo sistema (klic KILL). Vsak proces ima pravico do neprekinjene uporabe procesorja N urinih prekinitev (če sam pred tem ne kliče kakega od podprogramov, ki mu spremeni stanje ali pa se ne zbudi kak proces z višjo prioriteto). Tak algoritem razporejanja se je izkazal kot zadovoljiv pri aplikacijah v realnem času, pri interaktivnem delu pa se ni izkazal ravno najbolje. Zato smo algoritem spremenili z uvedbo dinamične prioritete, tako da je podoben razporejevalniku na VAX VMS /VMS82/. Po končanem čakanju na zmogljivosti razporejevalnik dvigne prioriteto interaktivnega programa za določeno vrednost (to vrednost določimo z uglaševanjem). Ob vsakem poskusu prerazporeditve procesov se prioriteta tega procesa spusti, dokler ne pride do svoje začetne (osnovne) prioritete. Računski procesi (compute bound) so ves čas na isti (osnovni) prioriteti.

3. UPORABA

Razporejevalnik smo temeljito testirali in tudi uporabili v realnih aplikacijah. Izkazalo se je, da je razporejevalnik dovolj robusten in univerzalen za uporabo na različnih področjih.

3.1 PROBLEM LAČNIH FILOZOFOV

Problem lačnih filozofov je prvič predstavil Hoare /HOA85/. Gre za omejeno število zmogljivosti. Rešitev, ki je predstavljena v dodatku B, preprečuje smrtni objem, vendar ne preprečuje stradanja. Kot je pokazalo testiranje, do stradanja ni prišlo, če smo opazovali sistem dovolj dolgo. Predstavljena rešitev uporablja monitor za zaščito kritičnega področja.

3.2 DELOVNE POSTAJE

Razporejevalnik smo preizkusili tudi v realni aplikaciji. Gre za računalnik združljiv z IBM-PC, na katerega je priključenih več delovnih postaj, ki asinhrono pošiljajo podatke po komunikacijskem kanalu računalniku, ki jih mora urediti, zapisati na disk in narediti obdelavo zbranih podatkov ter jo izpisati na zaslon. Aktivnosti posameznih postaj so med seboj neodvisne. Izkazalo se je, da je razporejevalnik dobro orodje, ki omogoča na preprost in pregleden način programiranje aplikacij za paralelno procesiranje.

4. NADALJNJE DELO

Razporejevalnik bomo preizkusili še na nekaterih problemih procesiranja v realnem času. Poiskali bomo najboljšo dolžino časovnega intervala neprekinjenega izvajanja procesa v okolju, ki zahteva izjemno hiter odziv in obenem hitro obdelavo podatkov. Poleg tega načrtujemo tudi primerjalno analizo različnih strategij razporejanja procesorskega časa (scheduling policy).

Razen tega nameravamo razširiti sistem z modulom, ki bo uvedel dodatne konstrukte za lažje paralelno programiranje (COBEGIN in COEND, REGION... kot na primer CC-Modula /COL87/ in jih priporoča /BSI87/) in nadaljevati delo s paralelnimi algoritmi.

5. ZAKLJUČEK

Predstavljeni sistem se je izkazal kot dobro programsko orodje za delo v večopravilnem okolju tako na področju poskusov s paralelnimi algoritmi kot na področju aplikacij za končne uporabnike. Sistem je dovolj robusten in kompakten tudi za težke pogoje dela v industriji.

LITERATURA

/HAN73/ Hansen,P.B.: Operating Systems Principles, Prentice Hall, 1973

/DIJ75/ Dijkstra,E.W.: Guarded Commands, Nondeterminancy and Formal Derivation of Programms, Comm.ACM, v.18 n.8, 1975

/WIR76/ Wirth N.: Algorithms + Data Structure = Programs, Prentice Hall, 1976

/HOL78/ Holt R.C., G.S.Graham, E.D.Lasowska, M.A.Scott : Structured Concurrent Programming with Operating Systems Applications; Addison Wessley, 1978

/VMS82/ Digital Equipment Corporation: VAX Software Handbook, DEC, 1982

/FIL84/ Filman,R.E., Friedman D.P.: Coordinated computing: Tools and Techniques for Distributed Software, McGraw Hill, 1984

/COM84/ Comer,Douglas: Operating System Design, the Xinu Approach; Prentice Hall, 1984

/WIR85/ Wirth,N.:Programming in Modula-2, Springer Verlag, 1985

/HOA85/ Hoare,C.A.R.: Communicating Sequential Processes, Prentice Hall, 1985

/LOG86/ Logitech: Modula-2/86, Logitech, 1986

/JON87/ Jones,G.: Programming in Occam, Prentice Hall, 1987

/BSI87/ BSI Modula-2 Working Group: Standard Concurrent Programming Facilities, N 116 Issue 3, 1987

/BRO87/ Brodnik A.: Programiranje z modulo-2, Informatica, 1987

/COL87/ Collado M.: A Modula-2 Implementation Of CSP, ACM Sigplan Notices, Vol. 22, N6, June 1987

DODATEK A (DEFINITON MODULE PROCESS)

```
DEFINITION MODULE PROCESS;

  FROM SYSTEM IMPORT ADDRESS;

  EXPORT QUALIFIED
    tProcState,    (* tip moznih stanj procesa *)
    PCREATE, PRESUME, PKILL, PSUSPEND, PSETPRIO, PGETPID,
    PGETSTAT, PGETCURR, PSLEEP, PRECEIVE, PSEND,
    tPid, tPrio, tSemaphore,
    cProcNameLen,
    tProcName, tProcStatus, tStatBlock,
    SCREATE, SINIT, SDELETE, SWAIT, SSIGNAL, SSTATUS, SGETSID,
    cSemNameLen,
    tSemName, tSemStatus,
    cPrio1,  cPrio2, cPrio3, cPrio4,  cPrio5, cPrio6, cPrio7;

  CONST
    cSemNameLen = 20;
    cProcNameLen = 20;          (* Process name lenghth *)

  TYPE
    tProcState = (PrCurr,       (* Process Current *)
                  PrReady,      (* Process Ready *)
                  PrSusp,       (* Process Suspended *)
                  PrWait,       (* Process Wait *)
                  PrSleep,      (* Process Sleeping *)
                  PrRec,        (* Process Receiving *)
                  PrFree        (* Process Slot in Table not Used *) );

    tProcName = ARRAY [0..cProcNameLen-1] OF CHAR;
    tProcStatus = (NOTOK,OK);
    tStatBlock = RECORD   (* undeveloped *)
                   status : CARDINAL
                 END;
    tPrio;                      (* process priority - type *)
    tPid;                       (* process - type *)
    tSemaphore;                 (* semaphore - type *)
    tSemName = ARRAY [0..cSemNameLen-1] OF CHAR;
    tSemStatus = (SNOTOK, SOK);

  VAR
    cPrio1, cPrio2, cPrio3, cPrio4, cPrio5, cPrio6, cPrio7 : tPrio;

  PROCEDURE PCREATE (process:PROC; envsize:CARDINAL; name:ARRAY OF CHAR;
                     VAR pid : tPid; prio: tPrio) : tProcStatus;
    (* Creates a new process with PrSusp status *)

  PROCEDURE PRESUME  (pid : tPid) : tProcStatus;
    (* Resumes a suspended process (PrSusp --> PrReady) *)

  PROCEDURE PKILL    (pid : tPid) : tProcStatus;
    (* Deletes a process and releases its resources (PrFree) *)

  PROCEDURE PSUSPEND (pid : tPid) : tProcStatus;
    (* Suspends a process (PrReady or PrCurr --> PrSusp) *)

  PROCEDURE PSETPRIO (pid : tPid; prio : tPrio) : tProcStatus;
    (* Change process priority *)

  PROCEDURE PGETPID  (name:ARRAY OF CHAR; VAR pid : tPid) : tProcStatus;
    (* returns process id (Pid) of process NAME *)

  PROCEDURE PGETSTAT (pid : tPid; VAR statblock:tStatBlock) : tProcStatus;
    (* returns process status informaton *)

  PROCEDURE PGETCURR () : tPid;
    (* returns process own process id *)

  PROCEDURE PSLEEP (time : CARDINAL):tProcStatus;
    (* Sleep for time ticks (PrCurr --> PrSleep) *)

  PROCEDURE PRECEIVE (VAR what:ADDRESS; VAR who:tPid): tProcStatus;
    (*  Receive  a message from anybody (conditional  PrCurr  --> PrRec) *)

  PROCEDURE PSEND (whom:tPid; what:ADDRESS) : tProcStatus;
    (* Send a message to a process *)

  PROCEDURE SCREATE (name:tSemName; VAR sem:tSemaphore; count:CARDINAL):
                     tSemStatus;
    (* Create a new sempahore and return its ID in sem *)
```

```
  PROCEDURE SINIT (sem:tSemaphore; count:CARDINAL) : tSemStatus;
    (* Init a Semaphore to initial value count *)

  PROCEDURE SDELETE (sem:tSemaphore) : tSemStatus;
    (* Delete a semaphore *)

  PROCEDURE SWAIT  (sem:tSemaphore) : tSemStatus;
    (* Wait for a semaphore (PrCurr --> PrWait) *)

  PROCEDURE SSIGNAL (sem:tSemaphore) : tSemStatus;
    (* Signal on semaphore sem *)

  PROCEDURE SSTATUS  (sem:tSemaphore) : tSemStatus;
    (* Get Semaphore status information *)

  PROCEDURE SGETSID (name:tSemName;VAR sem:tSemaphore;count:CARDINAL)
                    : tSemStatus;
    (* Get the ID of semaphore name *)

END PROCESS.
```

DODATEK B (DINING PHILOSOPHERS)

```
MODULE SPAGHETTI;
FROM PROCESS IMPORT PCREATE, PRESUME, PSLEEP, PSUSPEND, PGETCURR,
                    PRECEIVE, PSEND, PKILL, PSETPRIO,
                    tPid, tProcStatus,
                    cPrio1, cPrio2, Port,
                    SCREATE,  SWAIT, SSIGNAL, SDELETE,
                    tSemName, tSemStatus, tSemaphore,
                    CURSOR, WRITEI, WRITE;
FROM InOut IMPORT WriteString, WriteCard, WriteLn, WriteHex;
FROM Break IMPORT EnableBreak;
FROM Strings IMPORT Concat;
FROM SYSTEM IMPORT GETREG, ADDRESS, WORD, BYTE, INBYTE, OUTBYTE, CODE;
FROM Keyboard IMPORT KeyPressed, Read;
FROM Devices IMPORT  SaveInterruptVector, RestoreInterruptVector;
FROM MyRandom IMPORT Random;


TYPE
  tPhil = [0..4];

VAR
  pid1,pid2,pid3,pid4,pid5 : tPid;
  procstat, status: tProcStatus;
  chn,I,J: INTEGER;
  w: WORD;
  konec : BOOLEAN;
  ch: CHAR;
  vec: ADDRESS;

MODULE FORKS [7];
IMPORT tSemaphore, tPhil, tProcStatus, tSemStatus, SWAIT, SCREATE, SSIGNAL,
       tSemName, Concat, PSLEEP;
EXPORT PICKUP, PUTDOWN;
  TYPE
      tNumOfFork = [0..2];

  VAR
      numOfForks : ARRAY tPhil OF tNumOfFork;
      ready : ARRAY tPhil OF tSemaphore;
      semstat : tSemStatus;
      i : INTEGER;
      tmp : tSemName;

PROCEDURE PICKUP (phil : tPhil);
  VAR
    left, right : tPhil;
BEGIN
  IF (numOfForks[phil] < 2) THEN
    semstat := SWAIT (ready[phil])
  END;
  right := phil;
  left  := (phil + 1) MOD 5;
  DEC(numOfForks[left]);
  DEC(numOfForks[right]);
END PICKUP;
```

```
PROCEDURE PUTDOWN (phil : tPhil);
  VAR
      left, right : tPhil;
BEGIN
  right := phil;
  left  := (phil + 1) MOD 5;
  INC(numOfForks[left]);
  INC(numOfForks[right]);
  IF numOfForks[left] = 2 THEN
    semstat := SSIGNAL (ready[left]);
  END;
  IF numOfForks[right] = 2 THEN
    semstat := SSIGNAL (ready[right]);
  END;
END PUTDOWN;

BEGIN  (* FORKS *)
  FOR i:= 0 TO 4 DO
    CASE i OF
      0 : tmp := 'Bacon' |
      1 : tmp := 'Sokrates' |
      2 : tmp := 'Aristoteles' |
      3 : tmp := 'Nietsche' |
      4 : tmp := 'Hegel'
    END (* case *);
    semstat := SCREATE (tmp, ready[i], 0);
    numOfForks[i] := 2;            (* forks initialization *)
  END;
END FORKS;

PROCEDURE First;
VAR
    timesEaten : CARDINAL;
BEGIN
  timesEaten := 0;
  LOOP
    PICKUP(0);
    WriteString ('Bacon eating ');
    WriteCard(timesEaten, 5);
    WriteLn;
    INC(timesEaten);
    PUTDOWN(0);
    procstat := PSLEEP (Random(10) );
  END;
END First;

PROCEDURE Second;
VAR
    timesEaten : CARDINAL;
BEGIN
  timesEaten := 1;
  LOOP
    PICKUP(1);
    WriteString ('Sokrates eating ');
    WriteCard(timesEaten, 5);
    WriteLn;
    INC(timesEaten);
    PUTDOWN(1);
    procstat := PSLEEP (Random(10) );
  END;
END Second;

PROCEDURE Third;
VAR
    timesEaten : CARDINAL;
BEGIN
  timesEaten := 1;
  LOOP
    PICKUP(2);
    WriteString ('Aristoteles eating ');
    WriteCard(timesEaten, 5);
    WriteLn;
    INC(timesEaten);
    PUTDOWN(2);
    procstat := PSLEEP (Random(10) );
  END;
END Third;
```

```
PROCEDURE Fourth;
VAR
    timesEaten : CARDINAL;
BEGIN
  timesEaten := 1;
  LOOP
    PICKUP(3);
    WriteString ('Nietsche eating ');
    WriteCard(timesEaten, 5);
    WriteLn;
    INC(timesEaten);
    PUTDOWN(3);
    procstat := PSLEEP (Random(10) );
  END;
END Fourth;

PROCEDURE Fifth;
VAR
    timesEaten : CARDINAL;
BEGIN
  timesEaten := 1;
  LOOP
    PICKUP(4);
    WriteString ('Hegel eating ');
    WriteCard(timesEaten, 5);
    WriteLn;
    INC(timesEaten);
    PUTDOWN(4);
    procstat := PSLEEP (Random(10) );
  END;
END Fifth;


BEGIN
  EnableBreak;
  status := PSETPRIO(PGETCURR(),cPrio2);
  status := PCREATE(First,800H,'Bacon',pid1,cPrio1);
  status := PCREATE(Second,800H,'Sokrates',pid2,cPrio1);
  status := PCREATE(Third,800H,'Aristoteles',pid3,cPrio1);
  status := PCREATE(Fourth,800H,'Nietsche',pid4,cPrio1);
  status := PCREATE(Fifth,800H,'Hegel',pid5,cPrio1);
  status := PRESUME(pid1);
  status := PRESUME(pid2);
  status := PRESUME(pid3);
  status := PRESUME(pid4);
  status := PRESUME(pid5);
  konec := FALSE;
  LOOP
    Port;
    status := PSLEEP(18);
    IF KeyPressed() THEN
      EXIT;
    END;
  END;  (* LOOP *)
  status := PKILL(PGETCURR());
  WriteString('END - Philosophers');
END SPAGHETTI.
```