Întrebare: cum am putea testa corectitudinea programului, în sensul de a verifica dacă într-adevăr programul creează în mod corect ierarhia de procese cerută conform enunțului problemei?

Răspuns: evident, ne vom folosi de informațiile afișate pe ecran în urma execuției programului, din care vom extrage relațiile de rudenie "tată -- fiu" și vom verifica că aceste relații de rudenie descriu exact ierarhia cerută în enunț (în cazul în care programul este corect), respectiv că nu o descriu (în cazul în care ați făcut greșeli logice, nu sintactice (!), în program).

Prin urmare, se pune întrebarea cum extragem relațiile de rudenie "tată -- fiu" și le "asamblăm" într-o ierarhie de procese?
Sunt mai multe moduri posibile de a face aceasta, pe care le vom detalia în cele ce urmează:

Metoda "vizuală":

Extragem vizual aceste relații de rudenie din mesajele afișate pe ecran și construim mintal ierarhia acestor relații.
Dezavantajul acestei metode: o putem aplica cu succes doar pentru ierarhii foarte simple, de genul celor de la exercițiile rezolvate **[N childs]** și **[A list of processes (v1 și v2)]**; pentru ierarhii ceva mai complexe, sunt șanse mari să greșim când facem această verificare mintală.

Metoda "manuală":

Extragem vizual aceste relații de rudenie din mesajele afișate pe ecran și desenăm manual, pe o foaie de hârtie, ierarhia acestor relații de rudenie. Mai precis, luăm o foaie de hârtie și scriem pe ea toate valorile numerice ale PID-urilor ce apar în mesajele afișate, trecând fiecare PID o singură dată pe hârtie (!), indiferent de numărul de apariții ale acestuia în mesajele afișate. Apoi, conectăm printr-o săgeată orientată fiecare pereche de PID-uri, despre care am văzut că sunt în relația părinte-->fiu în vreunul din mesajele afișate.
"Desenul" care rezultă astfel este un graf orientat, și probabil estetic nu va fi foarte plăcut (unele arce se vor intersecta, etc.). Oricum, din acest desen ar trebui totuși să puteți să vă dați seama ce fel de formă are graful orientat obținut... e.g., este un arbore binar complet? ; este un "lanț"? ; este un arbore `k`-ar cu `N` nivele? etc.
Dacă graful obținut are forma ierarhiei de procese cerute în enunțul problemei, înseamnă că programul este corect, i.e. deci nu ați făcut greșeli în program. Iar dacă graful obținut arată altfel decât ierarhia cerută, diferențele dintre graf și ierarhia cerută v-ar putea ajuta să descoperiți ce greșeli ați făcut în program!
Dezavantajul acestei metode: pentru un număr mare de procese și o ierarhie "stufoasă" (i.e, nodurile au mai mulți fii), există probabilitatea să faceți greșeli când realizați desenul pe foaia de hârtie prin procedura descrisă mai sus.

Cum am putea elimina acest neajuns al metodei "manuale"? *Răspuns*: prin automatizarea ei (!), realizată prin metoda descrisă mai jos.

Metoda automatizată, bazată pe log-uri:

Ideea generală este aceea de prelucrare automată, după terminarea execuției programului, a informațiilor ce descriu anumiți parametri ai acelei execuții, parametri ce au fost salvați într-un fișier de tip log.
În cazul nostru, parametrii sunt informațiile de rudenie părinte-->fiu, iar fișierul de log folosit este "scrierea de mesaje pe ecran". Iar tipul de prelucrare a acestor informații, în cazul nostru, constă în construcția acelui graf orientat ce capturează ierarhia acestor relații de rudenie, graf descris la metoda anterioară.

Cum am putea face această construcție a grafului într-o manieră automatizată, i.e. fără a mai desena manual graful respectiv pe o foaie de hârtie, pentru a înlătura probabilitatea de a face greșeli când desenăm manual graful?
O posibilitate de a face automatizat această construcție a grafului, este aceea de a scrie un script bash care să "interpreteze" mesajele afișate la execuția programului și să "deseneze pe ecran", în mod automatizat, ierarhia de procese create.
Ideea scriptului este să ne folosim de sistemul de fișiere pentru a "simula" ierarhia relațiilor de rudenie printr-un subarbore de fișiere construit în mod adecvat, după care se va lista, într-o manieră "grafică" sugestivă, acel subarbore de fișiere.
Scriptul respectiv, denumit `fork_p2ls.sh`, este următorul: Show / Hide the script

```
#!/bin/bash
mkdir fork_p1tmp

echo "Elementele de intrare:"
while read proc
do
    if ( echo $proc | grep "^[0-9]\{1,\}-[0-9]\{1,\}$" )
    then
        mkdir fork_p1tmp/$proc
    else
        echo "Intrare incorecta : $proc"
    fi
done

echo -e "\nIerahia de procese este:"
change=0
until [ $change -eq 1 ]
do
    for proc in fork_p1tmp/*
    do
        ppid=$( echo $proc | cut -d- -f2 )
        pdir=$( find fork_p1tmp -name "$ppid-*" )
        if [ "$pdir" != "" ]
        then
            mv $proc $pdir
            change=1
        fi
    done
done

for proc in $( find fork_p1tmp/* )
do
    for secv in $( echo $proc | tr / " " | cut -d" " -f3- )
    do
        echo -n "   |"
    done
    echo -n "   |-"
    basename $proc | cut -d- -f1
done

rm -r fork_p1tmp
```

Explicații: scriptul primește un set de perechi de valori numerice de forma "PID-PPID" și desenează ierarhia de procese determinată de acel set.
De exemplu, pentru perechile date ca intrare:

```
1800-21632
1803-1800
1804-1803
1806-1804
1805-1804
1802-1800
1807-1803
1808-1802
1810-1807
1811-1808
1812-1802
1814-1812
1815-1808
1813-1812
1809-1807
```

scriptul va afișa următoarea ierarhie de procese:

```
Ierahia de procese este:
  |-1800
  |   |-1802
  |   |   |-1812
  |   |   |   |-1814
  |   |   |   |-1813
  |   |   |-1808
  |   |   |   |-1811
  |   |   |   |-1815
  |   |-1803
  |   |   |-1807
  |   |   |   |-1810
  |   |   |   |-1809
  |   |   |-1804
  |   |   |   |-1805
  |   |   |   |-1806
```

deci, cu alte cuvinte, se folosește o identare de la stânga la dreapta pentru a reprezenta fiii fiecărui proces (similar cu outputul uneltei "Process Explorer" utilizate în Windows pentru a afișa informații despre procesele active în sistem).

Observație: pentru a ușura considerabil scrierea scriptului, partea de "interpretare" a textelor mesajelor afișate pe ecran de programul dvs., în sensul extragerii din mesaje a perechilor de forma "PID-PPID", nu a fost automatizată prin acest script (!). Și atunci, aveți două soluții la îndemână:
i) ori extrageți manual / vizual perechile respective și le furnizați manual ca input scriptului;
ii) ori puteți să modificați mesajele tipărite de programul dvs. astfel încât să afișeze doar texte de forma "PID-PPID", și atunci este suficient să rulați scriptul înlănțuit cu programul dvs., i.e. cu comanda:
prompt> `./MyProgramWithForks.exe | ./fork_p2ls.sh`
și astfel nu veți fi nevoiți, când rulați scriptul, să introduceți manual perechile extrase din mesajele afișate la execuția programului dvs.
(*Notă*: dacă ați scris programul dvs. astfel încât să afișeze mesaje mai lungi, de genul: "`Procesul <i>, cu PID-ul:<X>, cu PID-ul parintelui:<Y>, ...`" , atunci fie modificați mesajele în program ca să resepcte formatul necesar scriptului, precum am spus mai sus, fie puteți să lăsați programul nemodificat, dar atunci va trebui să prelucrați manual output-ul programului, cu comenzi cut adecvate, pentru a extrage din el formatul "PID-PPID" necesar scriptului.)

La pasul final (ce este similar cu cel de la metoda anterioară) al acestei metode automatizate parțial, vă rămâne ca sarcină, de executat manual, "recunoașterea" formei pe care o are ierarhia afișată de script, i.e. dacă are sau nu forma ierarhiei de procese cerute în enunțul problemei?

Metoda automatizată, bazată pe "inspectarea" în timp real a execuției programului, cu comanda pstree:
O altă metodă automatizată are la bază ideea de a "observa" în timp real procesele create de programul dvs., folosind comanda `pstree`, care afișează procesele active din sistem, printr-o reprezentare cu identare de la stânga la dreapta, care reflectă ierarhia proceselor.
Pentru a restricționa outptul comenzii doar la ierarhia proceselor care ne interesează, trebuie să apelați comanda `pstree` cu următorii parametri:
prompt> `pstree -p value`
unde *value* este PID-ul shell-ului ce rulează în fereastra terminal în care vă testați programul. Sau, dacă testați programul pe serverul fenrir, și aveți o singură sesiune deschisă, mai puteți specifica, drept *value*, și *username*-ul dvs. de pe server.

Mai rămâne un aspect de precizat: comanda `pstree` capturează, practic, un *snapshot* al stării sistemului (i.e., al proceselor active din sistem) de la momentul execuției sale.
Prin urmare, comanda `pstree` trebuie rulată imediat după ce ați pornit execuția programului dvs., nu după ce s-a terminat deja. Aceasta o puteți face astfel:
prompt> `./MyProgramWithForks.exe > MyOutput.txt & pstree -p value`
Dar nu e suficient să faceți doar atât, ci mai trebuie și să vă asigurați că toate procesele create de programul dvs. **încă** sunt *alive* (adică, nu s-au terminat deja) în momentul când se execută comanda pstree. Pentru aceasta, trebuie să vă asigurați că toate procesele sunt create cât mai repede, chiar la începutul execuției programului dvs., și că durata de viață a **fiecărui** proces este suficient de lungă (i.e., de ordinul a câtorva secunde), pentru a nu se termina înainte de a se executa comanda pstree.
Iar aceasta o puteți realiza introducând, de exemplu, apeluri `sleep(5);`, într-o manieră "inteligentă", în programul dvs. (!)
Adică, trebuie să vă asigurați că apelul `sleep` se va executa chiar la finalul execuției fiecăruia dintre procesele create la execuția programului dvs., și nu mai înainte de final, ori, mai precis, nu înainte ca respectivul proces să fi terminat de creat toți fiii pe care trebuie să-i creeze conform cerințelor din problema respectivă. De aceea am spus mai sus că apelurile `sleep` trebuie introduse într-o manieră "inteligentă" în programul dvs., în sensul că, pentru a decide unde să le introduceți în program, trebuie să analizați NU doar **structura sintactică** a programului, ci și, mai ales, **"flow"-ul de execuție** al programului. Acesta va fi însă un "flow" ce se ramifică arborescent (!), datorită apelurilor `fork`, deci nu va fi un "flow" liniar (ca în cazul programelor secvențiale pe care erați obișnuiți să le scrieți (și să le analizați) la disciplinele întâlnite anterior acestui semestru), ceea ce va spori dificultatea alegerii corecte a poziției (sau pozițiilor) din program unde trebuie inserate aceste apeluri `sleep`.

Recomandare: încercați să aplicați (măcar) una dintre cele două metode automatizate descrise mai sus (pe care anume? pe care doriți dvs. ; sau, încercați-le pe amândouă...), pentru a verifica corectitudinea tuturor programelor din prima parte a acestui laborator (atât a celor din exerictiile rezolvate, cât și a programelor pe care le veți scrie pentru a rezolva problemele propuse spre rezolvare).