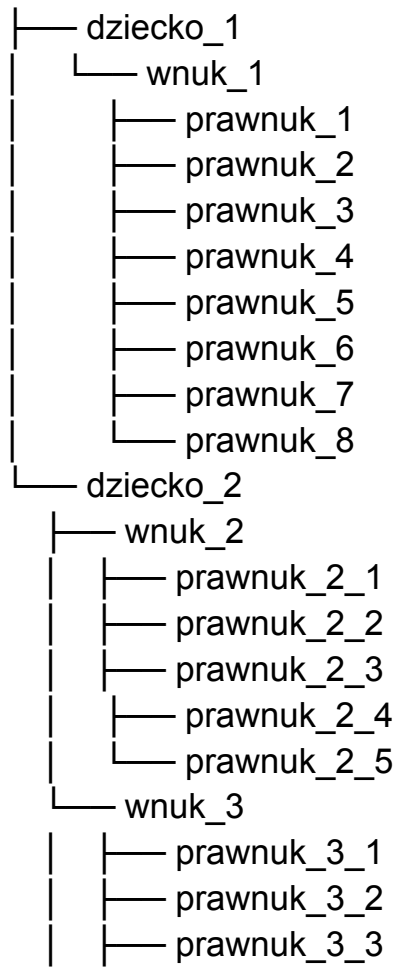
Tobiasz Kownacki Soi lab1 protokół z testów

Przypomnienie koncepcji testów:
Zadanie 1

Testująca funkcja będzie przyjmować dodatkowy jeden argument, nazwijmy go x. W funkcji stworzę 2 procesy potomne nazwijmy je dziecko_1 i dziecko_2. Dziecko_1 stworzy kolejne 2*x dzieci. Natomiast dziecko_2 stworzy x dzieci, a następnie każde dziecko stworzy kolejne x dzieci. W ten sposób dziecko_1 będzie miał ogólnie 2*x potomków a dziecko_2 x+x^2. Funkcja na samym początku wypisuję aktualną liczbę procesów na podstawie funkcji a następnie wypisuję wszystkie procesy czyli sumę (aktualne+2+2*x+x+x^2).

Zadanie 2
Testująca funkcja będzie przyjmować tylko argumenty z zadania. W funkcji stworzę taki schemat procesów jak widać poniżej:



```
|   ├── prawnuk_3_4
|   └── prawnuk_3_4
```

Minix na samym rozpoczęciu systemu ma 7 procesów, także dałem dużo nowych procesów aby być pewnym, że funkcja zwróciła pid nowego procesu a nie systemowego minixa.
Dla N = 1, funkcja powinna zwrócić 8 i pid wnuka_1 bo wnuk_1 ma 8 potomków na 1 pokoleniu w dół. Dla N = 2, funkcja powinna zwrócić 12 i pid dziecka_2, bo ma 2 potomków w pierwszym pokoleniu i 8 w drugim co daje w sumie 8. Dla N = 3 powinna zwrócić 23 i pid rodzica.

Testy:

Zadanie 1.
Wzór na liczbę procesów wygenerowanych przez program testowy: (2+2*x+x+x^2).
Na początku aby przetestować samą ilość generowanych procesów ustawiam A=1 i B=100, a parametr x=1.

```
* ./test1 1 100 1
Number of init children before fork: 7
Max children between 1 and 100: 13
Pid of process with max children between 1 and 100: 0
```

Liczba dzieci procesu init przed stworzeniem procesów potomnych wynosi 7, a po juz 13. Odejmując te liczby otrzymujemy liczbę 6, co jest zgodne z powyższym wzorem. Numer Pid wynosi 0, czyli wskazuje na Inita, który jest ojcem wszystkich procesów.

```
# ./test1 1 100 2
Number of init children before fork: 7
Max children between 1 and 100: 19
Pid of process with max children between 1 and 100: 0
```

Gdy ustawimy x=2, również liczba stworzonych procesów potomna(19-7=12) jest zgodna z wzorem.
Także z tych dwóch testów wynika, że wywołanie systemowe poprawnie zlicza liczbę wszystkich procesów potomnych.

Następnie przetestuje wywołanie systemowe zmniejszając zakres A i B. Ustawiam A=4, B=8 i x=4.

```
# ./test1 4 10 4
Number of init children before fork: 7
Max children between 4 and 10: 8
Pid of process with max children between 4 and 10: 182
#
```

Program zwraca mi liczbę 8, co oznacza, że tym maksymalnym procesem jest proces dziecko_1, który miał stworzyć 2*x procesów.

Teraz sprawdzę czy odpowiednio dopasowując parametry, program zwróci mi liczbę dzieci posiadaną przez proces dziecko_2, czyli x+x^2.

```
# ./test1 18  22 4
Number of init children before fork: 7
Max children between 18 and 22: 20
Pid of process with max children between 18 and 22: 222
#
```

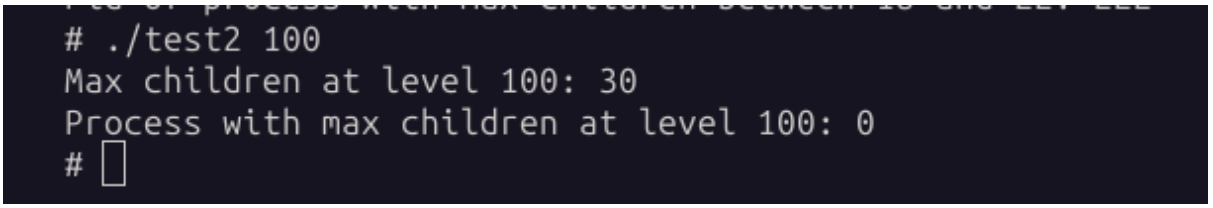
Program zwrócił liczbę 20, czyli liczbę dzieci posiadaną przez dziecko_2.

Zadanie 2.

```
# ./test2 1
Max children at level 1: 8
Process with max children at level 1: 91
* ./test2 2
Max children at level 2: 12
Process with max children at level 2: 124
* ./test2 3
Max children at level 3: 23
Process with max children at level 3: 137
*
```

Uruchomienie programu testowego z parametrami N=1, 2, 3 zwraca nam takie wartości jakie były opisane w koncepcji. Ilość zwracanych procesów jest zgodna z drzewkiem procesów potomnych.

Podając bardzo wysokie N, program powinien nam zwrócić numer pid Inita, czyli zero bo jest on ojcem wszystkich procesów.



Wywołanie systemowe zwraca poprawny numer PID. Liczba dzieci również jest zgodna, ponieważ przed utworzeniem nowych procesów przez program testujący init ma 7 dzieci (co pokazałem przy testowaniu zadania 1), a sam program testujący tworzy ich 23.