System Programing

Assignment 1

이름: 이찬하

학번: 201811097

2021년 10월 23일

<문제 정의>

ku_ff.c 라는 하나의 파일에 다음과 같은 동작을 하는 프로그램을 작성해야 한다.

A. 프로그램은 실행시 3가지 인자 를 입력 받는다.
   1. 첫번째 인자: 찾을 대상 범위의 시작값
   2. 두번째 인자: 찾을 대상 범위의 끝값
   3. 세번재 인자: 찾는데 동원될 프로세스의 수

B. ku_ps_input.h라는 헤더파일이 실행환경에 존재한다. 해당 파일에는 다음에 대한 정보가 있다.
   1. 탐색의 대상이 되는 배열
   2. 탐색 대상의 크기

C. 실행된 프로세스는 대상의 크기를 세번째 인자로 나눈 크기 만큼의 범위에서 조건에 맞는 숫자를 검색하는 자식 프로세스를 만들고, 그들이 찾은 결과를 취합한다.

D. 모든 자식프로세스의 결과들과 자신이 맡은 범위에 대한 결과를 취합한 뒤 표준 출력에 그 결과를 출력한다

<프로그램 진행 순서>

0. 유저는 적절히 ku_ff.c를 컴파일하여 프로그램을 실행한다.

1. 메인함수에서 유저의 입력을 분석한다.

다음 중 하나에 해당할 경우 프로그램이 즉시 종료된다.

a. 입력된 인자의 개수가 3개가 아닐때
b. 찾을 범위의 시작값($1)이 찾을 범위의 끝값($2)보다 클 때
c. 동원될 프로세스의 개수($3)가 1보다 작을 때

2. sigprocmask()를 사용하여 이후의 동작들이 SIGCHILD에 대해 방해받지 않도록 한다.

3. [$3 - 1] **회 반복하는 while 문을 실행한다.**

while문 내부

1. fork()를 실행하여 자식프로세스를 생성한다.
   A. 자식프로세스일 경우
   1. 2.번에서 언급된 while문이 얼마나 실행되었는지 기록하는 변수를 통해 자신이 검색해야 할 범위를 추정한다.

2. 해당 범위에서 주어진 조건에 맞는 숫자들의 갯수를 파악한다. (annalyseArr 함수를 사용)
3. sendMessage 함수를 사용해 sisV 메시지 전송방식으로 메시지 큐에 파악한 갯수를 전송한다.
4. child의 exit(0);

B. 부모 프로세스일 경우 이곳(while 내부)에서 따로 하는 동작은 없다.

4. ($3 - 1) **회 반복하는 while 문을 실행한다.**

이전 while문에서 자식들은 모두 exit을 만나도록 구성 되었기에 이곳의 while 문은 자식에겐 unreachable 하다.

1. sigsuspend()를 사용하여 SIGCHILD를 전송받을 수 있는 상태로 전환하고, 종되된 child를 하나 reaping한다.
2. child가 전송한 메시지 하나를 getMessage함수로 꺼내 result에 더한다.

5. 자신에게 배당된 범위에 대해 조건에 맞는 숫자의 개수를 구하여 result 에 더한다.

6. result를 표준출력에 출력한다.

7. return(0) (부모 프로세스 종료)

<실행 결과 분석>

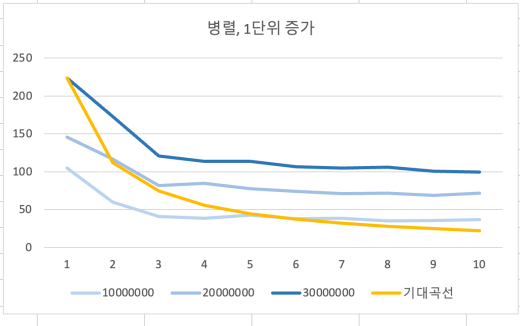
헤더 파일을 원하는 크기만큼 생성해 주는 프로그램과 그것을 자동으로 테스트 해주는 코드를 작성하여 실행을 분석하였다.

만든 프로그램 깃허브 링크: https://github.com/lee-chanah/Homeworks_public/tree/master/system_programing_HW/HW1

2. 프로세스 개수 증가량이 1일 경우

	배열 크기			
프로세스 수	10000000	20000000	30000000	기대곡선
1	105	146	224	224
2	60	117	173	112
3	41	82	121	74 2/3
4	39	85	114	56
5	43	78	114	44 4/5
6	38	74	107	37 1/3
7	39	71	105	32
8	35	72	106	28
9	36	69	101	24 8/9
10	37	72	100	22 2/5

단위: millisec



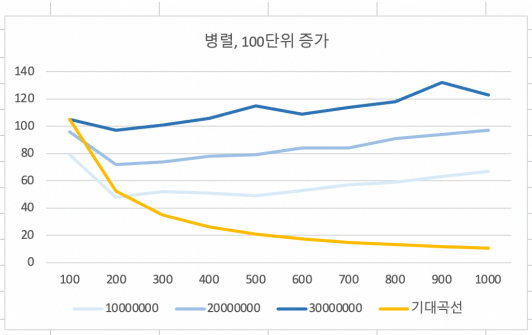
프로세스가 1개씩 증가하도록 실행 프로그램에 입력할 경우 프로세스 개수가 증가할 수록 소요시간이 대체로 감소하는 형태를 나타낸다. 노란색 기대곡선은 자식프로세스가 동시에 실행되며, 생성, 수확 메시지 수발신 소요시간이 0일 경우 이상적인 소요시간에 대한 곡선이다 (시간 = 105 / (프로세스 개수)). 기대곡선과 비교해 소요시간이

큰 이유는 생성한 모든 프로세스가 동시에 실행될 수 없고, fork와 reap, 메시지 전송에 소요되는 시간의 영향으로 생각된다.

1. 100개 단위로 프로세스가 증가할 경우

	배열 크기			
프로세스 수	10000000	20000000	30000000	기대곡선
100	79	96	105	105
200	48	72	97	52 1/2
300	52	74	101	35
400	51	78	106	26 1/4
500	49	79	115	21
600	53	84	109	17 1/2
700	57	84	114	15
800	59	91	118	13 1/8
900	63	94	132	11 2/3
1000	67	97	123	10 1/2

단위: millisec



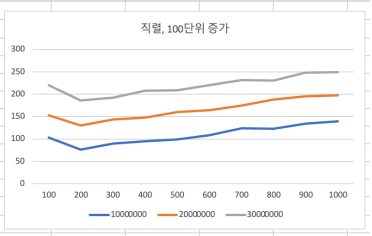
프로세스가 100개씩 증가할 경우 실행 시간이 초회 이후 지속적으로 증가하는 것을 확인할 수 있다. 프로세스가 병렬로 실행되며 동시에 연산하여 단축한 전체 실행 시간보다 fork와 reaping, 그리고 메시지 전송, 수신으로 인해 증가하는 시간의 영향이 더 크다는 것을 확인할 수 있다.

3. 추가 실험 -프로세스의 직렬 실행과 비교

100개 단위로 프로세스의 개수가 증가할 때, 소요시간이 덩달아 증가하는 것을 확인했다. 그렇다면 직렬로 즉, 프로세스의 생성과 종료가 번갈아 진행되어 동시에 실행되는 자식 프로세스가 없도록 구성할 경우 그 소요시간을 알아보고 병렬 실행과의 차이를 파악해보겠다. (직렬 실행의 코드는 위에서 언급 했던 주소에 있다.)

	배열 크기		
프로세스 수	10000000	20000000	30000000
100	103	153	220
200	76	130	186
300	90	144	192
400	95	148	208
500	99	160	209
600	109	165	220
700	124	175	232
800	123	188	231
900	134	196	249
1000	140	198	250

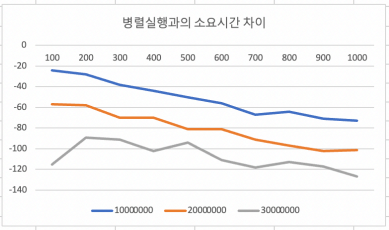
단위: millisec



좌측의 표와 그래프는 프로세스의 직렬 실행시 사용된 프로세스에 따른 소요시간에 대한 그래프이다. 모든 실행에서 그 소요시간이 병렬 실행보다 크다는 것을 확인할 수 있다.

그렇다면 이번에는 병렬실행과의 시간 차이를 그래프로 나타내 보겠다.

	10000000	20000000	30000000
100	-24	-57	-115
200	-28	-58	-89
300	-38	-70	-91
400	-44	-70	-102
500	-50	-81	-94
600	-56	-81	-111
700	-67	-91	-118
800	-64	-97	-113
900	-71	-102	-117
1000	-73	-101	-127



배열의 크기가 3천만인 경우 다소 기울기가 일정하지 않지만, 대체로 3경우 모두 시간차이가 증가함을 보이고 있다. (y축의 값이 음수임에 주의)

이 그래프가 나타내는 것은 프로세스 유지비용(fork, reap, mqSend, mqGet 등) 을 고려 하지 않았을 때, 여러 개의 프로세스가 구역을 소분하여 탐색할 경우 얼마나 시간이 감소하는 지에 대해 나타낸다고 할 수 있다. 여기에 몇개의 프로세스가 동시에 실행될 수 있는지, 등 아직 모르는, 배우지 않은 내용 들에 대해선 고려하지 않았다.

Function description

Function Name	Arguments	Description
main	int argc	함수 진행을 총괄 **argc**: 전달받은 인자의 개수 **argv**: 인자들이 저장된 문자열 배열
	char **argv	
	return int	
sendMessage	int msg	전달받은 msg를 sisV 메시지큐 전송 방식으로 메시지 큐에 전달 **msg**: 전달할 메시지
	return void	
getMessage	return int	sisV 메시지 수취방식으로 메시지 큐에서 메시지 하나를 가져와 담긴 data 반환
analyseArr	int indexFrom	배열에서 찾는 구간과 찾을 숫자의 범위에 대해 전달받아 해당 구간 내의 숫자들을 검사. **indexFrom**: 탐색구간 시작값 **indexTo**: 탐색구간 끝값 **findFrom**: 찾을 숫자 시작값 **findTo**: 찾을 숫자 끝값
	int indexTo	
	int findFrom	
	int findTo	
	return int	